SAPIENZA
UNIVERSITÀ DI ROMA

Pure Transformers Can Be Powerful Hypergraph Learners

Department of Computer, Control and Management Engineering
Corso di Laurea Magistrale in Artificial Intelligence and Robotics

Candidate
Peng Kai
ID number 1947951

Thesis Advisor
Dr. Giovanni Trappolini

Co-Advisor
Prof. Fabrizio Silvestri

Academic Year 2022/2023

Pure Transformers Can Be Powerful Hypergraph Learners
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: peng.1947951@studenti.uniroma1.it

Abstract

Graph convolutional networks (GCNs) and Hypergraph convolutional networks (HGCNs) have gained significant attention in recent years. However, the widely used message-passing methods in GCNs and HGCNs often suffer from oversmoothing issues. Additionally, combining transformers with graphs (including hypergraphs) has shown promise, but modified transformer approaches have their own limitations. To address these problems and expand to hypergraph area, we propose Tokenized Hypergraph Transformer(TokenHGT), a novel approach that processes hypergraphs as tokens and leverages a pure transformer architecture. By treating hypergraphs as tokens, tokenHGT effectively mitigates oversmoothing problems commonly encountered in message-passing methods. Moreover, it overcomes the limitations associated with modified transformers. To evaluate the performance of tokenHGT, we conduct experiments on text classification datasets. Specifically, we test our algorithm on the R8 and R52 datasets, the results demonstrate that tokenHGT achieves competitive results, and outperforming all message-passing methods in terms of classification accuracy in the MR dataset. TokenHGT offers a promising approach for improving hypergraph-based classification tasks. The proposed framework opens up possibilities for enhanced performance and scalability in various applications involving hypergraph data.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Giovanni Trappolini and my co-supervisor Fabrizio Silvestri for their invaluable guidance, support, and encouragement throughout the course of this research.

I would also like to extend my heartfelt appreciation to my parents for their unwavering love, encouragement, and continuous support. Their belief in me has been a constant source of motivation, and I am grateful for their sacrifices and understanding.

Finally, thanks to myself on this journey.

Contents

Chapter 1

Introduction

This master thesis presents Tokenized Hypergraph Transformer(TokenHGT), a standard transformer[45] model for processing hypergraph datasets. Unlike common methods that rely on message-passing, TokenHGT organizes hypergraphs as tokens within the standard transformer structure. By doing so, it overcomes the problems inherited from message-passing techniques such as oversmoothing[31][36][4], and as confirmed by experimental results. The proposed approach enhances the scalability of hypergraph processing, offering a promising alternative in complex data analysis.

1.1 From Graphs to Hypergraphs: Extending TokenGT to Explore New Research Horizons

The ever-increasing complexity of data representation and analysis demands innovative approaches to capture intricate relationships within structured data. Graph neural networks(GNNs)[39] have gained significant attention in recent years for their ability to model relational data represented as graphs. By leveraging node features and graph structures, GNNs can capture complex relationships and make predictions on graph-structured data.

However, the traditional graph model comes with certain limitations that have led researchers to explore and develop hypergraphs. One significant limitation is the inability of graphs to capture complex relationships involving more than two nodes simultaneously. This restricts their ability to represent higher-order interactions and dependencies in many real-world scenarios. Moreover, graphs struggle to handle heterogeneous data, where nodes and relationships have diverse attributes and types. To overcome these limitations, hypergraphs have emerged as an extension of the graph model. Hypergraphs allow for hyperedges, which can connect any number of nodes, enabling the representation of complex relationships and dependencies more accurately. Hypergraphs also excel in capturing group relationships, as they naturally represent subsets or clusters of nodes as a single entity. Additionally, hypergraphs offer the ability to handle heterogeneous data effectively, making them suitable for modeling real-world systems with diverse attributes. These advantages make hypergraphs an intriguing and worthwhile research area. By delving into hypergraph theory and exploring its practical applications, researchers can unlock new insights and develop innovative solutions to complex problems. Hypergraphs have already found applications in fields such as computer science, social networks, bioinformatics, and knowledge representation. The research on hypergraphs has the potential to enhance data analysis techniques, improve decision-making processes, and provide a deeper understanding of interconnected systems. Therefore, investigating hypergraphs is worth pursuing to overcome the limitations of graphs and advance our ability to model, analyze, and solve complex real-world problems.

One powerful approach for harnessing the capabilities of hypergraphs is through hypergraph neural networks(HGNNs)[15], which extend the GNN framework to handle hypergraphs more effectively. HGNNs incorporate hyperedge information and enable more comprehensive modeling of hypergraph data.

Nonetheless, Li, Q. et al.[31] prove that the graph convolution is actually a special form of Laplacian smoothing and it also brings potential concerns of over-smoothing. Chen, C. et al.[4] extends the over-smoothing analysis beyond linear cases and investigates the over-smoothing effect in the broader context of GNN architectures, the aim is to provide a more comprehensive understanding of the over-smoothing phenomenon in GNNs. Oono, K. et al.[36] focuses on analyzing the phenomenon of over-smoothing and provides theoretical conditions to understand when GCNs experience information loss as the number of layers approaches infinity. Building upon this theory, a principled guideline is proposed to determine the appropriate scale of weights in GNNs.

The transformer[45] architecture leverages self-attention mechanisms to capture global dependencies and efficiently process sequential data. It has emerged as a powerful model in various domains, and has also shown promising results in graph and hypergraph analysis.

Despite its powerful self-attention mechanism, the global self-attention approach in transformers cannot inherently capture the structural characteristics of graphs. Consequently, researchers have proposed graph-specific architectural modifications to address this limitation. But it may impose limitations on the versatility of the models.

Tokenized Graph Transformer(TokenGT)[26] has successfully addressed two significant problems in the graph domain: the issue of "over-smoothing" resulting from message-passing methods and the limitations imposed by graph-specific architectural modifications on the versatility of models. However, these problems have not yet been solved in the context of hypergraphs, motivating our research to explore their potential, this endeavor is driven by the recognition of the value and significance of researching hypergraphs as mentioned earlier. Given the aforementioned significance of researching hypergraphs, our goal is to extend TokenGT to the field of hypergraphs to solve the limitations of message-passing and graph-specific structural modifications in the hypergraph field, and provide an optional method for processing hypergraphs.

To achieve our goal, we face several challenges and have developed corresponding approaches. Firstly, hypergraphs require a different approach to Laplacian eigendecomposition compared to graphs. We have found a hypergraph Laplacian eigendecomposition formula 2.6 & 2.7 to address this challenge. Secondly, hyperedges in hypergraphs can contain varying numbers of nodes, unlike the fixed two nodes in graph edges. This prevents us from using TokenGT's edge identifier to align eigenvectors. To overcome this challenge, we directly employ feature fusion, which is explained in detail in Chapter 3.1.1. Additionally, our proposed approach operates at the graph level, necessitating a suitable graph-level hypergraph dataset. Since such datasets are challenging to find, we tackle this problem by converting text data into hypergraphs, as outlined in Chapter 4.1.

In summary, we have successfully addressed the above challenges and successfully achieved our goal, which also is our main contribution.

1.2 Structure of the thesis

In this subsection, we present the structure of the thesis.

In Chapter 1, we provide an overview of the research topic and highlight the motivation, goal and challenges. We present the structure of the thesis, outlining the key sections and their contributions.

In Chapter 2, we present an overview of key concepts in graph analysis and neural networks. We begin by introducing graphs and graph neural networks (GNNs). Next, we explore the concepts of hypergraphs and hypergraph neural networks (HGNNs), which offer a more flexible representation for complex relationships. Additionally, we discuss the relevance of transformers in processing both graphs and hypergraphs. These concepts form the foundation for the subsequent chapters of our thesis.

In Chapter 3, we focus on the proposal and implementation of the Tokenized Hypergraph Transformer (TokenHGT), which consists of three main components: Node Identifier, Special Token, and Main Transformer. Then, we introduce the implementation details of tokenHGT.

The Chapter 4 focuses on evaluating the proposed TokenHGT framework through various datasets.Additionally, we analyze the accuracy of TokenHGT and compare it against other existing methods.

The Chapter 5 discusses the implications of our results and their potential impact on future research and applications.

Chapter 2

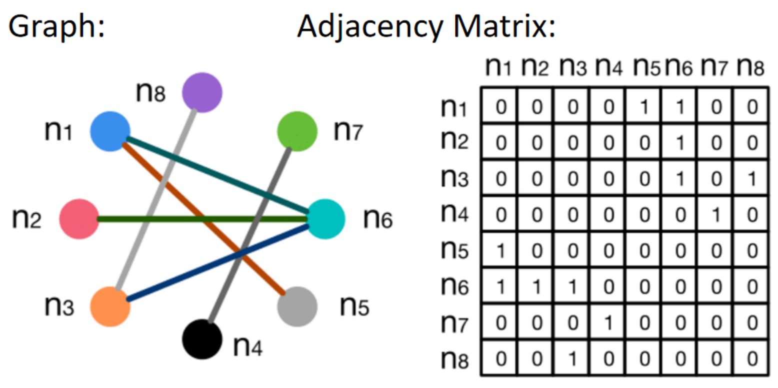
Background and Related Works

2.1 Graph and GNN

2.1.1 Graph

Graphs are fundamental mathematical structures that represent relationships between objects. In a graph, objects are represented as nodes or vertices, while the relationships between them are depicted as edges. Graphs are widely used in various fields, including computer science, social networks, transportation networks, and biological systems. Graphs provide a powerful framework for modeling and analyzing complex systems. They allow us to study the connectivity, dependencies, and interactions between entities. Graph-based algorithms and techniques enable us to extract meaningful insights[47][12][10], detect patterns[35][18], and solve problems such as graph clustering[29][16], community detection[50][2], shortest path finding[44][42], and recommendation systems[22][7].

Additionally, the graph can be visualized using a node-link diagram, where vertices are represented as nodes or circles, and edges are depicted as lines or arcs connecting the nodes. The layout and arrangement of the nodes and edges in the diagram provide insights into the connectivity and structure of the graph. An example of a graph, as illustrated in Figure 2.1.



	n1	n2	n3	n4	n5	n6	n7	n8
n1	0	0	0	0	1	1	0	0
n2	0	0	0	0	0	1	0	0
n3	0	0	0	0	0	1	0	1
n4	0	0	0	0	0	0	1	0
n5	1	0	0	0	0	0	0	0
n6	1	1	1	0	0	0	0	0
n7	0	0	0	1	0	0	0	0
n8	0	0	1	0	0	0	0	0

Figure 2.1. A visual representation of the relationships between nodes and edges in a graph. In this example, we have a set of nodes $\nu = \{v_1, ..., v_8\}$ and a set of edges $\varepsilon = \{e_1, ..., e_6\}$. Each vertex represents a distinct entity, while the edges signify the connections or interactions between these entities.

A graph is defined as $G = (\nu, \varepsilon)$, which includes n nodes $\nu = \{v_1, ..., v_n\}$ and m edges $\varepsilon = \{e_1, ..., e_m\}$, associated with features $X^\nu \in \mathbb{R}^{n \times C}$ and $X^\varepsilon \in \mathbb{R}^{m \times C}$. The graph G can be denoted by a $n \times n$ adjacency matrix A, with entries defined as:

$$A(i,j) = \begin{cases} 1, & \text{if } (v_i, v_j) \in e \\ 0, & \text{if } (v_i, v_j) \notin e \end{cases} \tag{2.1}$$

Additionally, in some cases, weighted graphs are considered, where the edges have associated weights. In such cases, the adjacency matrix can contain the weight values instead of just 0s and 1s, the adjacency matrix is defined as follows:

$$A(i,j) = \begin{cases} W_{i,j}, & \text{if } (v_i, v_j) \in e \\ 0, & \text{if } (v_i, v_j) \notin e \end{cases} \tag{2.2}$$

The graph Laplacian is a mathematical operator that is derived from the adjacency matrix of a graph. It is a fundamental concept in graph theory and plays a crucial role in various graph-based algorithms and analyses. It is a powerful tool that aids in understanding and analyzing the properties and behavior of graphs, making it an essential concept in graph theory and its applications.

For the graph $G = (\nu, \varepsilon)$, the graph Laplacian matrix L of G is defined as

$L = D - A$, where D is the degree matrix and A is the adjacency matrix of the graph. The graph Laplacian matrix L provides important information about the graph's structure and properties. It captures the relationships between vertices and their neighboring vertices, enabling various graph analysis techniques such as spectral graph theory[41], graph clustering, graph partitioning, and solving graph-based optimization problems.

The normalized Laplacian matrix eigendecomposition[26], also referred to as spectral decomposition, can be defined as follows:

$$\Delta = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U^T \Lambda U \tag{2.3}$$

Where I is identity matrix, Λ represents the eigenvalues, and U corresponds to the eigenvectors[14]. The eigenvectors associated with the normalized Laplacian matrix are orthogonal to each other. The eigenvectors of graph capture unique structural patterns within the graph. These eigenvectors correspond to different frequencies or modes of oscillation, with the lower eigenvectors representing global structural information and the higher eigenvectors capturing more localized features[6]. The eigenvectors are orthogonal to each other, meaning they are linearly independent and do not share any common structural characteristics. By examining and analyzing these eigenvectors, we can gain insights into the underlying connectivity, community structure, and other important properties of the graph.

Understanding graphs Laplacian matrix and eigenvectors is essential for comprehending the subsequent chapters of this thesis, where we apply these concepts to tackle hypergraph analysis and explore the potential of tokenized hypergraph for standard transformers.

2.1.2 Graph Neural Networks(GNNs)

Graph Neural Networks (GNNs) are a class of deep learning models specifically designed to process and analyze graph-structured data. Unlike traditional neural networks that operate on grid-like data, such as images or sequences, GNNs are

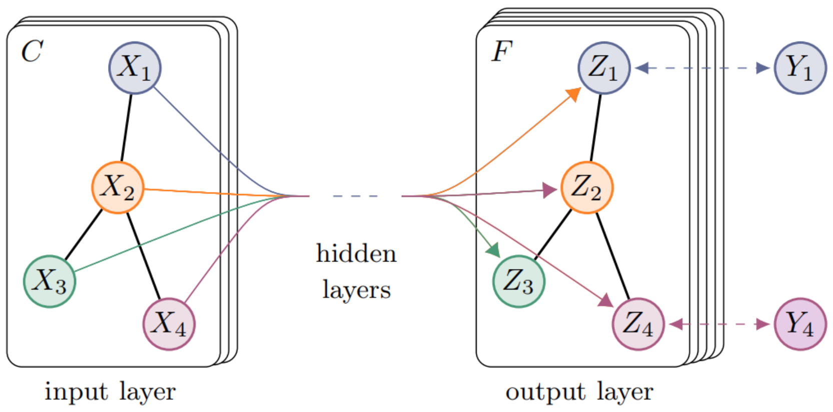


Figure 2.2. This is a schematic representation of a graph neural network (GNN).

capable of incorporating and leveraging the relational information present in graphs. Figure 2.2 is an GNN example.

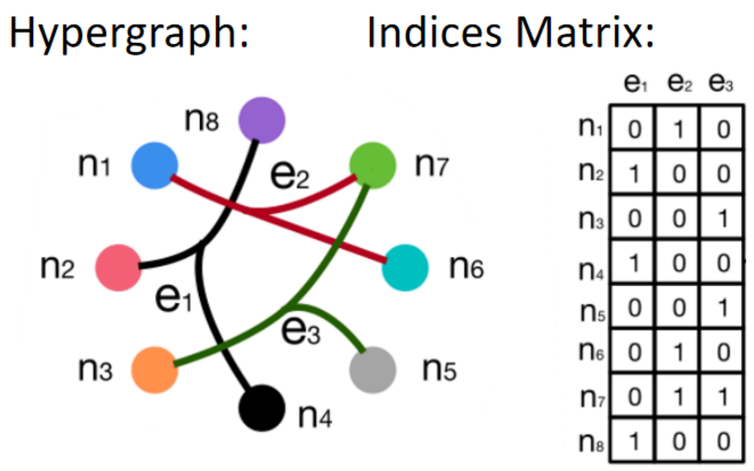
GNNs operate by aggregating and transforming information from neighboring nodes in the graph to update the representation of each node. This process is typically performed iteratively, allowing nodes to gather and propagate information across the graph. By combining local node features with information from neighboring nodes, GNNs can capture complex relationships and dependencies within the graph. A multi-layer Graph Convolutional Network (GCN)[27] employs a layer-wise propagation rule, which operates as follows:

$$H^{l+1} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.4}$$

where $\widetilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function. $H^{(l)} \in \mathbb{R}^{n \times C}$ is the matrix of activations in the l^{th} layer, $H^{(0)} = X^{\nu}$. As mentioned before, the graph convolution is actually a special form of Laplacian smoothing[31].

GNNs have shown great success in various graph-related tasks, such as node classification, link prediction, and graph-level prediction. They have been applied in diverse domains, including social networks[33][34], recommendation systems[48][17], chemistry[5][52], and computer vision[37][43].

The versatility and effectiveness of GNNs make them a powerful tool for understanding and analyzing graph-structured data. By learning and exploiting the underlying structure and relationships of the graph, GNNs offer promising avenues for solving a wide range of real-world problems.



	e_1	e_2	e_3
n_1	0	1	0
n_2	1	0	0
n_3	0	0	1
n_4	1	0	0
n_5	0	0	1
n_6	0	1	0
n_7	0	1	1
n_8	1	0	0

Figure 2.3. In this example, we have eight nodes labeled as $n_1, ..., n_8$. The hyperedges, denoted as e_1, e_2, e_3, connect different subsets of nodes. The indices matrix of the hypergraph is a binary matrix that indicates the presence or absence of nodes in each hyperedge. Rows in the matrix correspond to nodes, while columns correspond to hyperedges.

2.2 Hypergraph and HGNN

2.2.1 Hypergraph

A hypergraph is a generalization of a graph where an edge, also known as a hyperedge, can connect more than two vertices. This extension allows for more flexible and expressive representations of relationships among elements. In a hypergraph, each hyperedge can connect any number of vertices, and a vertex can participate in multiple hyperedges simultaneously. Compared to traditional graphs, hypergraphs offer several advantages. They can capture richer interactions among elements, provide a more natural representation of heterogeneous data, and enable a more expressive modeling of complex relationships. Additionally, hypergraphs allow for higher-order computations and offer greater flexibility in data representation. Hypergraphs find applications in various domains, including data mining, machine learning, social networks, recommendation systems[10], and knowledge representation[19]. They are particularly useful when dealing with complex and higher-order relationships among entities. Figure 2.3 illustrates an example of a hypergraph along with its corresponding indices matrix.

A hypergraph, denoted as $G = (\nu, \varepsilon)$, consists of a set of nodes ν and a set of hyperedges ε, which includes n nodes $\nu = \{v_1, ..., v_n\}$ and m hyperedges $\varepsilon = \{e_1, ..., e_m\}$, associated with features $X^{\nu} \in \mathbb{R}^{n \times C}$ and $X^{\varepsilon} \in \mathbb{R}^{m \times C}$. Unlike in traditional graphs where an edge connects exactly two vertices, a hyperedge in a hypergraph can connect any number of vertices. To represent a hypergraph mathematically, we can use an incidence matrix H, which is an $n \times m$ matrix, where n is the number of vertices and m is the number of hyperedges. The entry $H(i, j)$ in the incidence matrix is defined as follows:

$$H(i,j) = \begin{cases} 1, & \text{if node i is incident to hyperedge j,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

In hypergraph analysis, a key concept is the hypergraph Laplacian matrix, denoted as L, which captures the structural properties of the hypergraph. The hypergraph Laplacian matrix plays a similar role as the graph Laplacian in traditional graphs, serving as a measure of connectivity and providing valuable information about the hypergraph structure. The normalized hypergraph Laplacian is defined as:

$$L = I - D_v^{-\frac{1}{2}} H W D_e^{-1} H^T D_v^{-\frac{1}{2}} \tag{2.6}$$

where I is the identity matrix of nodes, D_v and D_e is the degree matrix of nodes and hyperedges, H is the indices matrix, W denotes the diagonal matrix containing the weights of hyperedges.

The hypergraph Laplacian matrix can be decomposed into its eigenvalues and eigenvectors, providing valuable insights into the structural properties of the hypergraph. The eigendecomposition formula for the hypergraph Laplacian matrix is as follows:

$$L = U^T \Lambda U \tag{2.7}$$

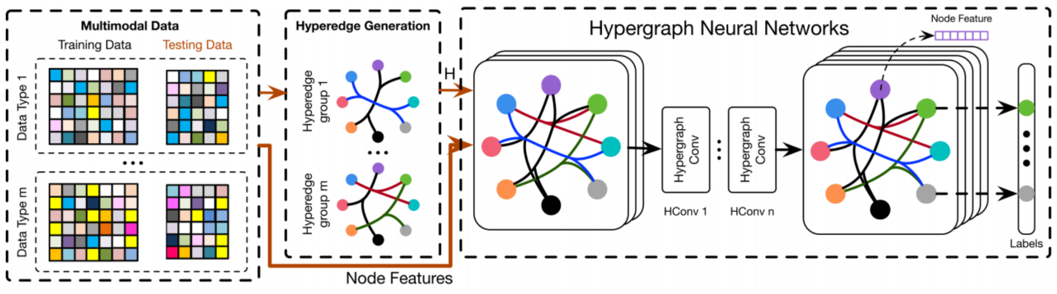


Figure 2.4. The hypergraph neural network(HGNN) framework.

In the context of hypergraph Laplacian eigendecomposition, the eigenvectors obtained from the decomposition play a crucial role in capturing important structural properties of the hypergraph. Similar to traditional graph Laplacian eigenvectors, these eigenvectors provide insights into the underlying connectivity patterns and information diffusion within the hypergraph. Each eigenvector corresponds to a specific eigenvalue and represents a different mode or frequency of oscillation within the hypergraph. The lower eigenvectors tend to capture global structural information, reflecting the overall organization and connectivity of the hypergraph. On the other hand, the higher eigenvectors capture more localized features, highlighting specific subsets or clusters of nodes within the hypergraph.

By analyzing the eigenvectors from the hypergraph Laplacian eigendecomposition, researchers can gain a deeper understanding of the hypergraph's characteristics, such as community structure, information flow, and hierarchical organization. These eigenvectors serve as valuable tools for exploring and analyzing complex relationships and dependencies present in hypergraphs.

2.2.2 Hypergraph Neural Networks(HGNNs)

To analyze and process hypergraphs, specialized techniques and models have been developed. Hypergraph Neural Networks (HGNNs)[15] extend Graph Neural Networks (GNNs) to handle hypergraph data. HGNNs incorporate hyperedge-level information into the learning process, enabling more effective modeling and reasoning about complex relationships. The Figure 2.4 showcases the HGNN framework.

In HGNN, the core operation is the hypergraph convolution, which incorporates information from both the nodes and hyperedges. Unlike graph convolution, which aggregates information only from neighboring nodes, hypergraph convolution considers the relationships among nodes within hyperedges, allowing for a more comprehensive understanding of the hypergraph structure. To perform hypergraph convolution, HGNN typically leverages a hypergraph Laplacian matrix, which encodes the connectivity and interactions among nodes and hyperedges. By decomposing the hypergraph Laplacian, HGNN can extract informative features and representations that are tailored to the specific characteristics of the hypergraph. The hypergraph convolution formula can be defined as follows:

$$X^{(l+1)} = \sigma(D_v^{-\frac{1}{2}} H W D_e^{-1} H^T D_v^{-\frac{1}{2}} X^{(l))} \Theta^{(l)}) \tag{2.8}$$

where $X^{(l)} \in \mathbb{R}^{n \times C}$ is the signal of hypergraph at l layer, $X^{(0)} = X^{\nu}$ and σ denotes the nonlinear activation function. The filter Θ is applied over the nodes in hypergraph to extract features. This hypergraph convolution formula incorporates information from the hypergraph structure, leveraging the hypergraph adjacency matrix and degree matrix to appropriately weight the node features. By applying this convolution operation iteratively across multiple layers, the hypergraph convolutional neural network can capture increasingly complex patterns and dependencies in the hypergraph data.

By leveraging the power of HGNNs, researchers have made significant advancements in tasks such as hypergraph classification, link prediction, and community detection. These approaches have demonstrated improved performance and a deeper understanding of hypergraph-structured data. As hypergraphs continue to gain attention and prove their efficacy in various domains, further research and development are expected to enhance our ability to analyze, model, and derive insights from complex, high-dimensional data.

2.3 Transformer

The transformer[45] is a powerful deep learning model that has revolutionized various domains, including natural language processing (NLP) and computer vision. It was first introduced in the context of machine translation,but one notable advantage of the transformer is its ability to handle long-range dependencies more effectively than traditional recurrent models. This makes it well-suited for tasks involving long sequences, such as machine translation, document summarization, and sentiment analysis. Additionally, the transformer introduces the concept of positional encoding, which injects positional information into the input sequence. This helps the model differentiate between elements based on their positions, enabling it to understand the order and structure of the input data. Figure 2.5 is an overview of the full architecture of transformer.

At the core of the transformer are two key components: the encoder and the decoder. The encoder takes an input sequence and generates a series of hidden representations, capturing the contextual information of each input element. The decoder, on the other hand, takes the encoder's outputs and generates the desired output sequence. Let's take a closer look at each component:

The transformer encoder takes an input sequence and produces a sequence of hidden representations. It consists of multiple layers, typically stacked on top of each other. Each layer in the encoder consists of two main sub-components: multi-head self-attention and feed-forward neural networks.

1. Multi-Head Self-Attention: This mechanism allows the encoder to weigh the importance of different elements in the input sequence. It computes attention scores for each element in the sequence, taking into account its relationships with other elements. The attention scores determine how much each element should contribute to the representation of other elements. Multi-head self-attention uses multiple attention heads to capture different types of dependencies and provides richer representations. The multi-head attention architecture, depicted in Figure 4.2

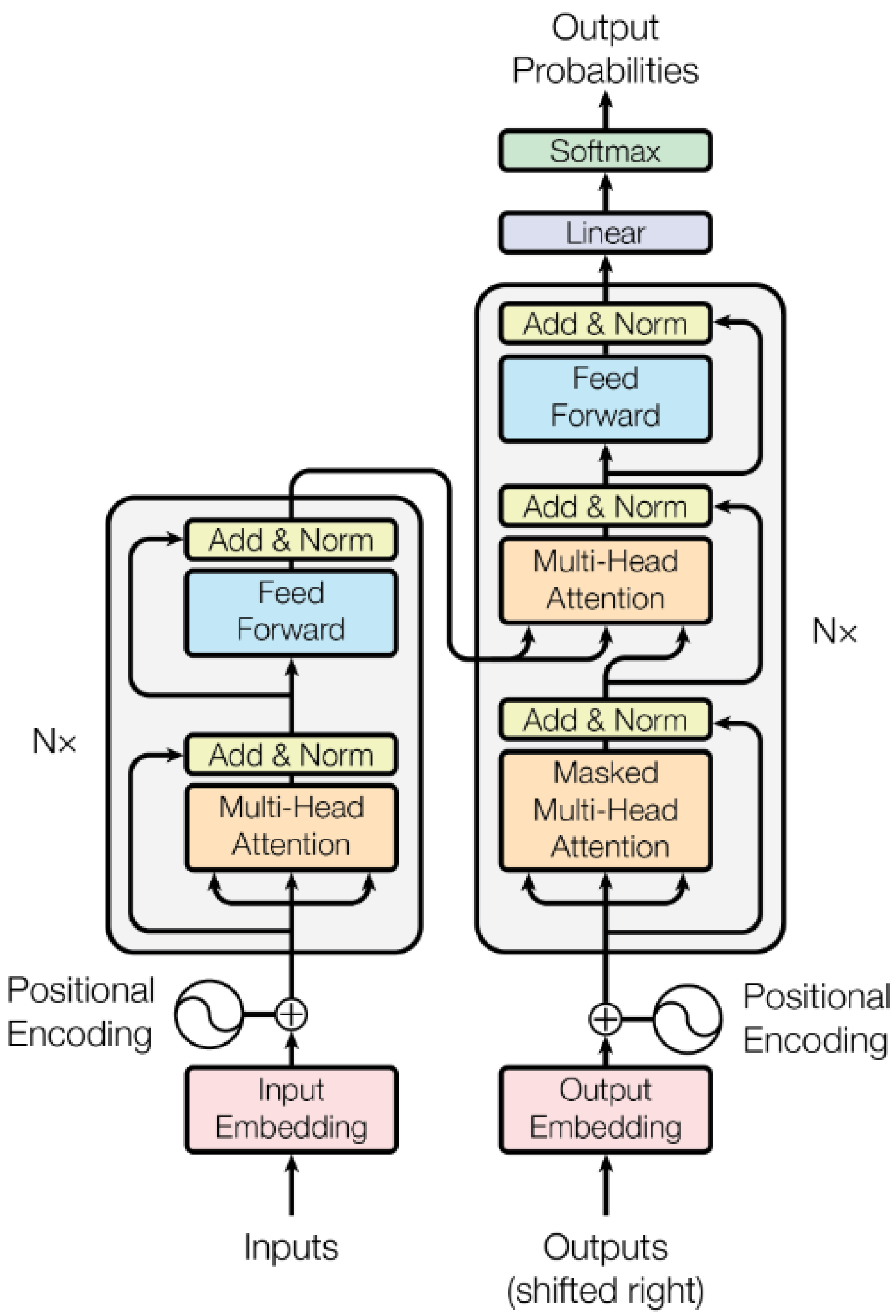


Figure 2.5. The Transformer - model architecture.

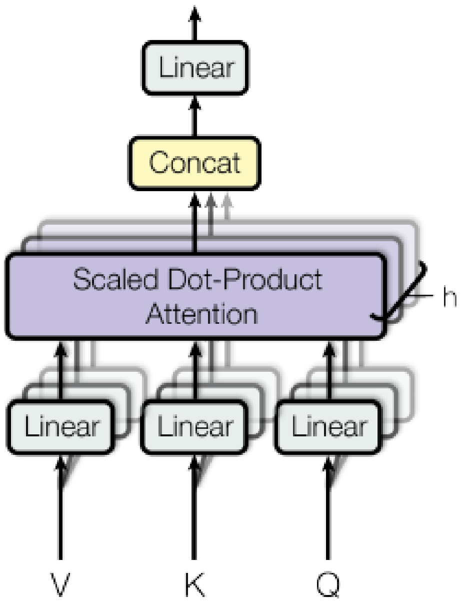


Figure 2.6. The multi-head attention is a crucial component of the transformer model that enables it to capture rich and diverse dependencies between elements in a sequence

2. Feed-Forward Neural Networks: After computing self-attention, the encoder applies a feed-forward neural network to each position in the sequence independently. This network processes the information from the attention step and applies non-linear transformations to generate more expressive representations. It helps capture complex patterns and dependencies within the sequence.

The transformer decoder takes the hidden representations from the encoder and generates the output sequence. It also consists of multiple layers, similar to the encoder. However, the decoder has an additional sub-component called the encoder-decoder attention.

1. Encoder-Decoder Attention: The decoder attends to the encoder's hidden representations to capture the relevant information for generating the output sequence. It computes attention scores between the decoder's current position and all

positions in the encoder's hidden representations. This allows the decoder to focus on the relevant parts of the input sequence while generating the output.

Both the encoder and decoder also incorporate positional encoding, which is a mechanism to embed positional information into the input sequence. This helps the model understand the order and structure of the input, as the transformer does not have inherent positional information like recurrent models.

A crucial aspect of the transformer is the self-attention mechanism. It allows the model to attend to different parts of the input sequence, giving more weight to relevant elements and suppressing irrelevant ones. This mechanism enables the transformer to model complex relationships and dependencies within the data. It plays a crucial role in capturing dependencies and relationships between elements in the sequence. Here's an overview of how the self-attention mechanism works:

1. Key, Query, and Value: In the self-attention mechanism, each element in the input sequence is associated with three learned vectors: the key vector, the query vector, and the value vector. These vectors are derived from the input embeddings and serve different purposes:

- Key: It represents the element's contextual information that will be used to compute attention scores with other elements.

- Query: It represents the element's position that will be attended to and receive information from other elements.

- Value: It contains the actual information associated with the element.

2. Attention Scores: To compute attention scores between elements, the self-attention mechanism uses a similarity measure between the query vector of one element and the key vectors of all other elements. This similarity measure is often

the dot product or a learned compatibility function. The result is a set of attention scores that quantify the relevance or importance of each element to the query element.

3. Attention Weights and Context Vector: The attention scores are normalized using a softmax function to obtain attention weights. These weights determine how much each element contributes to the representation of the query element. Elements with higher attention weights have a greater influence on the final representation. The attention weights are then used to compute a weighted sum of the value vectors, resulting in a context vector.

2.3.1 Transformer with Graph

In the context of graph data, researchers have explored integrating transformers to enhance the representation and modeling capabilities of GNNs. By combining the strengths of transformers' global contextual understanding with GNNs' ability to capture local node information, these transformer-GNN hybrid models have demonstrated improved performance in tasks such as node classification, graph classification, and link prediction. The transformer's attention mechanism allows it to capture relational information among nodes in a graph, enabling more effective and expressive graph representations.

Here are some references that explore the combination of transformer with graphs:

1. **TokenGT**: TokenGT[26] explores the potential of using standard Transformers for graph learning tasks, both theoretically and in practical applications. It demonstrates that by treating nodes and edges in a graph as independent tokens and incorporating token embeddings, we can effectively apply Transformers to graph data. This approach, which refer to as Tokenized Graph Transformer (TokenGT), yields promising results and is theoretically proven to be at least as expressive as an invariant graph network (2-IGN) composed of equivariant linear layers. The overview of tokenGT is depicted in Figure 2.7.

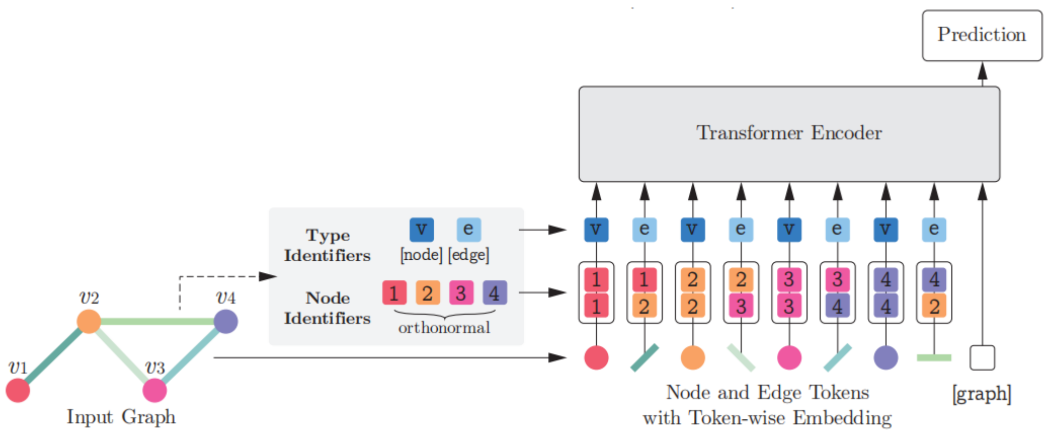


Figure 2.7. Overview of Tokenized Graph Transformer (TokenGT).

In TokenGT, each node and edge in the input graph is treated as an independent token. To enhance the model's ability to understand the graph structure, additional information in the form of orthonormal node identifiers and trainable type identifiers is incorporated. For graph-level predictions, TokenGT adopts a common practice where an extra trainable token, often referred to as the [graph] token, is included. This token captures the overall information of the graph and is used to make predictions at the graph level.

To validate the practical performance of TokenGT, the authors conducted experiments on a large-scale graph dataset, PCQM4Mv2. The results demonstrate that TokenGT outperforms GNN baselines, showcasing its effectiveness in graph learning tasks. Additionally, TokenGT achieves competitive results when compared to Transformer variants that incorporate sophisticated graph-specific inductive bias.

Our Tokenized Hypergraph Transformer(TokenHGT) model draws inspiration from the Tokenized Graph Transformer (TokenGT) approach. The details of tokenHGT will be introduced in Chapter 3.

2. **Graphomer**: The Transformer architecture has emerged as a powerful framework in various domains, such as natural language processing and computer vision. However, its performance on graph-level prediction tasks has not been

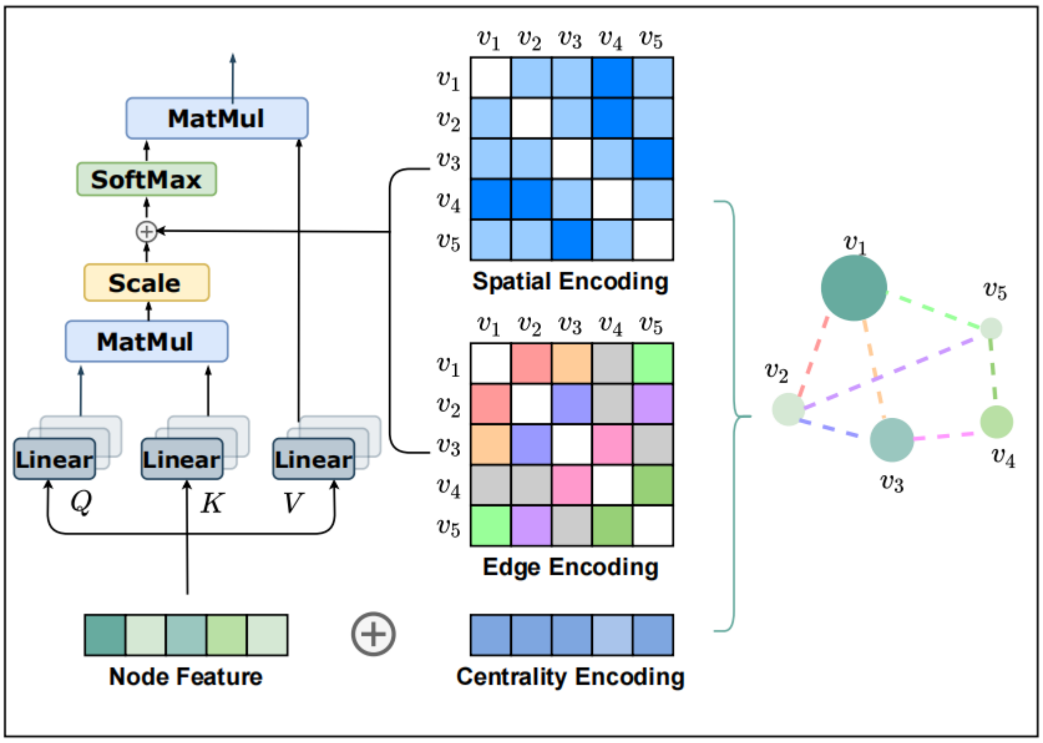


Figure 2.8. It provides a visual representation of the centrality encoding, spatial encoding, and edge encoding techniques used in Graphormer.

competitive compared to mainstream Graph Neural Network (GNN) variants. This discrepancy raises questions about how Transformers can effectively learn representations for graphs. [53] address this problem by introducing Graphormer, a novel approach that builds upon the standard Transformer architecture and achieves impressive results on a wide range of graph representation learning tasks, including the recent OGB Large-Scale Challenge. An illustration of encoding methods as shown in Figure 2.8.

The authors' key insight in leveraging Transformers for graph learning lies in the effective encoding of structural information within the model. they introduce three structural encoding methods in Graphormer. The first method is Centrality Encoding, which captures the importance of nodes in the graph. By incorporating centrality measures, such as node degree or betweenness centrality, Graphormer can effectively attend to the most relevant nodes in the graph. The second method is Spatial Encoding, which utilizes the shortest path distance between nodes as a demonstration. This distance is encoded as a bias term in the softmax attention mechanism, enabling the model to capture spatial dependencies accurately. The third method is Edge Encoding, which computes an average of dot products between edge features and learnable embeddings along the shortest path. This encoding captures important edge-level information and incorporates it into the attention module.

To further understand the capabilities of Graphormer, the authors mathematically characterize its expressive power. And demonstrate that by employing this structural encoding methods, Graphormer can encompass many popular GNN variants as special cases. This finding highlights the flexibility and versatility of Graphormer in capturing various graph structures and modeling graph-structured data.

The introduction of three novel graph structural encodings in Graphormer has yielded impressive performance on various benchmark datasets. These promising results indicate the potential of Graphormer in graph representation learning. However, several challenges still need to be addressed in order to further enhance its capabilities.

One significant challenge is the quadratic complexity of the self-attention module, which limits the scalability of Graphormer on large graphs. Future research efforts should focus on developing efficient variants of Graphormer that can handle graphs with a larger number of nodes and edges. By addressing this challenge, Graphormer can be applied to a wider range of real-world graph datasets.

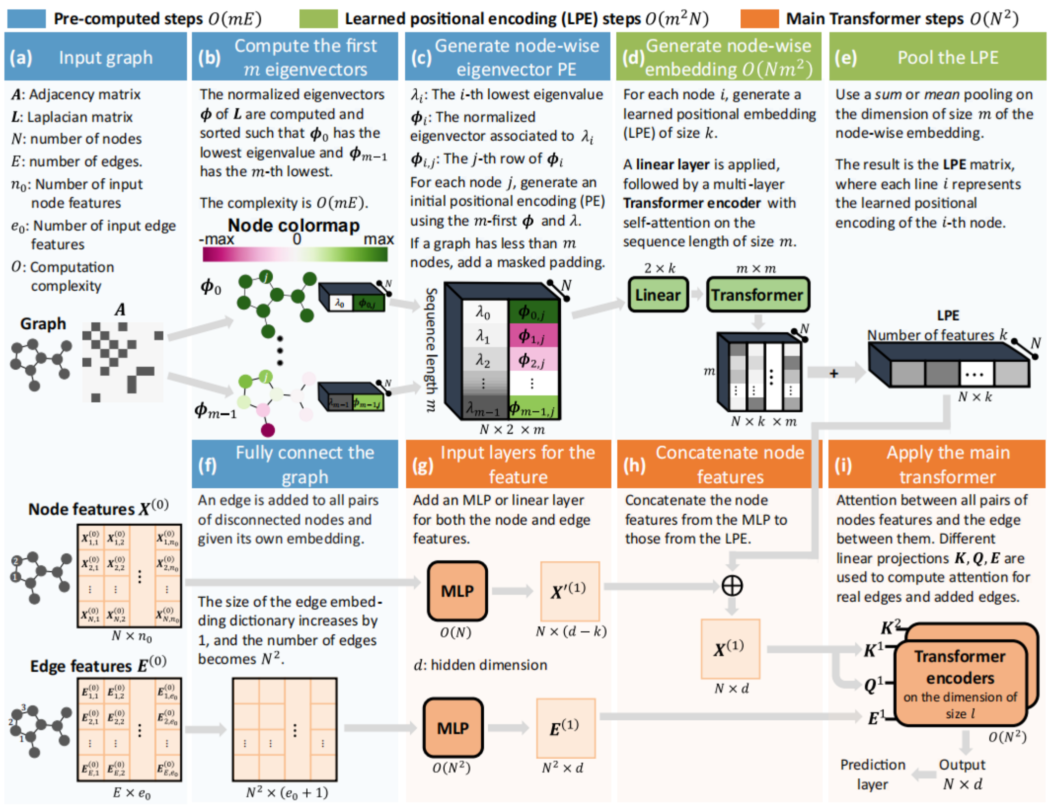
Additionally, the performance of Graphormer can be further improved by leveraging domain-specific knowledge to design graph-specific encodings. By incorporating domain expertise into the encoding process, Graphormer can capture important graph structural properties and enhance its representation learning capabilities for specific types of graphs.

Furthermore, an effective graph sampling strategy is desirable for extracting node representations using Graphormer. Developing a sampling technique that captures the most informative nodes while maintaining computational efficiency would enable the application of Graphormer to large-scale graphs.

These challenges and avenues for future research highlight the potential for further advancements in Graphormer. By addressing the scalability issue, incorporating domain knowledge, and developing efficient graph sampling strategies, Graphormer can continue to evolve as a powerful tool for graph representation learning and make significant contributions to the field.

3. **SAN**: The Transformer architecture has achieved remarkable success in sequence processing, but its application to graphs has been limited due to challenges in defining positions accurately. In this study, [28] introduce the Spectral Attention Network (SAN), which utilizes a learned positional encoding (LPE) to leverage the full Laplacian spectrum and determine the position of each node in a graph. By incorporating LPE into the node features and employing a fully-connected Transformer, the model demonstrates theoretical power in distinguishing graphs and effectively detecting similar sub-structures based on their resonance. Additionally, unlike most Graph Neural Networks (GNNs), the fully-connected Transformer avoids the issue of over-squashing, which can lead to an information bottleneck.

   This work focuses on the application of Transformer architectures in graph representation learning, and makes significant contributions through the development of novel learnable positional encoding methods rooted in spectral graph theory. The proposed positional encoding technique, along with the resulting SAN architecture, addresses theoretical limitations observed in previous graph


Pre-computed steps $O(mE)$
Learned positional encoding (LPE) steps $O(m^2N)$
Main Transformer steps $O(N^2)$
(a) Input graph
(b) Compute the first m eigenvectors
(c) Generate node-wise eigenvector PE
(d) Generate node-wise embedding $O(Nm^2)$
(e) Pool the LPE
(f) Fully connect the graph
(g) Input layers for the feature
(h) Concatenate node features
(i) Apply the main transformer
Graph
A
Node colormap
-max
0
max
ϕ_0
ϕ_{m-1}
j
N
λ_0 $\phi_{0,j}$
λ_{m-1} $\phi_{m-1,j}$
Sequence length m
λ_0 λ_1 λ_2 λ_{m-1}
$\phi_{0,j}$ $\phi_{1,j}$ $\phi_{2,j}$ $\phi_{m-1,j}$
$N \times 2 \times m$
$2 \times k$
$m \times m$
Linear
Transformer
m
$N \times k \times m$
+
LPE
Number of features k
$N \times k$
Node features $X^{(0)}$
$X^{(0)}_{1,1}$ $X^{(0)}_{1,2}$ $X^{(0)}_{1,n_0}$
$X^{(0)}_{2,1}$ $X^{(0)}_{2,2}$ $X^{(0)}_{2,n_0}$
$X^{(0)}_{N,1}$ $X^{(0)}_{N,2}$ $X^{(0)}_{N,n_0}$
$N \times n_0$
Edge features $E^{(0)}$
$E^{(0)}_{1,1}$ $E^{(0)}_{1,2}$ $E^{(0)}_{1,e_0}$
$E^{(0)}_{2,1}$ $E^{(0)}_{2,2}$ $E^{(0)}_{2,e_0}$
$E^{(0)}_{E,1}$ $E^{(0)}_{E,2}$ $E^{(0)}_{E,e_0}$
$E \times e_0$
$N^2 \times (e_0 + 1)$
MLP
$O(N)$
$X'^{(1)}$
$N \times (d-k)$
MLP
$O(N^2)$
$E^{(1)}$
$N^2 \times d$
$\oplus$
$X^{(1)}$
$N \times d$
K^2
K^1
Q^1
E^1
Transformer encoders
on the dimension of size l
$O(N^2)$
Prediction layer
Output
$N \times d$

A: Adjacency matrix
L: Laplacian matrix
N: number of nodes
E: number of edges.
n_0: Number of input node features
e_0: Number of input edge features
O: Computation complexity

The normalized eigenvectors ϕ of L are computed and sorted such that ϕ_0 has the lowest eigenvalue and ϕ_{m-1} has the m-th lowest.

The complexity is $O(mE)$.

λ_i: The i-th lowest eigenvalue
ϕ_i: The normalized eigenvector associated to λ_i
$\phi_{i,j}$: The j-th row of ϕ_i
For each node j, generate an initial positional encoding (PE) using the m-first ϕ and λ.
If a graph has less than m nodes, add a masked padding.

For each node i, generate a learned positional embedding (LPE) of size k.

A **linear layer** is applied, followed by a multi-layer **Transformer encoder** with self-attention on the sequence length of size m.

Use a *sum* or *mean* pooling on the dimension of size m of the node-wise embedding.

The result is the **LPE** matrix, where each line i represents the learned positional encoding of the i-th node.

An edge is added to all pairs of disconnected nodes and given its own embedding.

The size of the edge embedding dictionary increases by 1, and the number of edges becomes N^2.

Add an MLP or linear layer for both the node and edge features.

d: hidden dimension

Concatenate the node features from the MLP to those from the LPE.

Attention between all pairs of nodes features and the edge between them. Different linear projections K, Q, E are used to compute attention for real edges and added edges.

Figure 2.9. The proposed SAN model extends the Transformer architecture to graphs by incorporating a node Learned Positional Encoding (LPE).

Transformer studies[13] and demonstrates superior expressive power compared to standard message-passing GNNs. Empirical results consistently show that employing full Transformer-style attention leads to improvements compared to equivalent sparse message-passing models. Figure 2.9 illustrates the SAN model with node LPE.

The SAN model also served as a motivation for the development of tokenHGT. Unlike in tokenGT, in tokenHGT, we refer to the concatenation of node and edge features with Laplacian features from the SAN model, instead of using addition. Through extensive experimentation, we have observed that in specific tasks, the concatenation of node and edge features with Laplacian features, as used in tokenHGT, consistently outperforms the addition operation employed in tokenGT.

2.3.2 Transformer with Hypergraph

Similarly, transformers have been extended to handle hypergraph data, which offers a more flexible representation than traditional graphs. Hypergraphs allow hyperedges to connect multiple nodes simultaneously, enabling a richer representation of complex relationships. By encoding hypergraphs and feeding them into transformers, researchers have developed transformer models specifically tailored for hypergraph processing. These models, known as Hypergraph Transformers (HG-Transformers), have shown promising results in tasks such as hypergraph-based topic modeling and deep text classification. Overall, the transformer architecture has demonstrated its versatility and effectiveness in graph and hypergraph analysis. By leveraging self-attention mechanisms, transformers can capture both global and local relationships in structured data, leading to improved performance in various tasks. The integration of transformers into graph and hypergraph models opens up new possibilities for more advanced and expressive analysis of complex relational data.

Here we present several methods based on the Hypergraph Transformer method.

1. **HyperGAT**: Text classification is a critical research area with wide-ranging applications in natural language processing. While graph neural networks (GNNs) have shown promising results in this domain, their practical performance can be compromised due to their limitations. GNNs struggle to capture high-order interactions between words and are inefficient when dealing with large datasets and new documents. To overcome these challenges, the authors propose a principled model called Hypergraph Attention Networks (HyperGAT)[9]. HyperGAT enhances expressive power while reducing computational consumption for text representation learning. It's a novel model designed for inductive text classification. HyperGAT addresses the limitations of existing graph neural networks (GNNs) by effectively capturing high-order interactions between words and efficiently handling large datasets. Figure 2.10 is the overview of HyperGAT

2. **Hypergraph Transformer**: Knowledge-based visual question answering

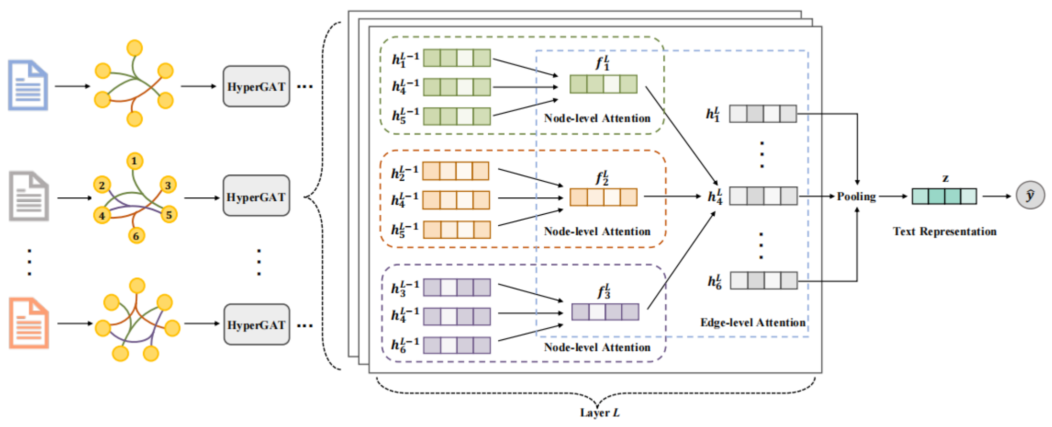


Figure 2.10. The overview of HyperGAT.

(QA) is a field that aims to answer questions by incorporating external knowledge beyond the image content. However, tackling complex questions that require multi-hop reasoning under weak supervision poses significant challenges. This is primarily due to the lack of supervision during the reasoning process and the need to capture high-order semantics of multi-hop knowledge facts. In this study, [19] propose the Hypergraph Transformer, a novel approach that utilizes hypergraph structures to encode and capture high-level semantics in both the question and knowledge base. By constructing question and knowledge hypergraphs, the model can infer answers by considering inter-associations between the hypergraphs and intra-associations within each hypergraph. The Hypergraph Transformer, a novel method that combines hypergraph structures and transformer-based attention mechanisms. This approach enables to encode multi-hop relationships and allows the model to learn to pay attention to crucial knowledge evidence for answering a given question. By explicitly constructing question and knowledge hypergraphs, the model can capture high-order semantics present in the question and each knowledge fact, effectively handling multi-hop relational knowledge facts. The Hypergraph Transformer leverages transformer-based attention mechanisms to perform hyperedge matching between the two hypergraphs, which is powerful for solving the multi-hop reasoning problem, as it enables the encoding of high-order semantics without length constraints and facilitates the learning of

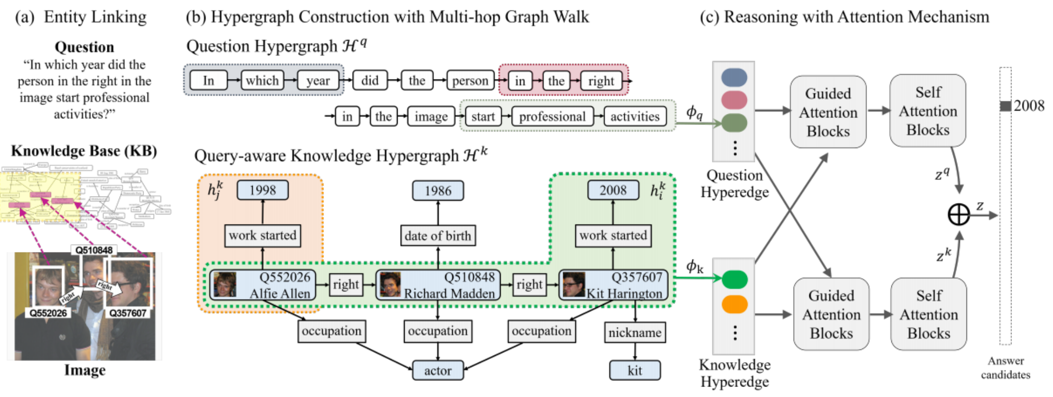

(a) Entity Linking
Question
"In which year did the person in the right in the image start professional activities?"
Knowledge Base (KB)
Image
(b) Hypergraph Construction with Multi-hop Graph Walk
Question Hypergraph $\mathcal{H}^q$
Query-aware Knowledge Hypergraph $\mathcal{H}^k$
(c) Reasoning with Attention Mechanism
Question Hyperedge
Knowledge Hyperedge
Guided Attention Blocks
Self Attention Blocks
Answer candidates
2008

Figure 2.11. The overview of Hypergraph Transformer.

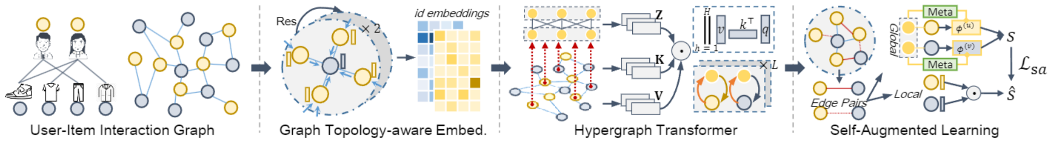


Figure 2.12. The overview of Self-Supervised Hypergraph Transformer framework.

cross-modal high-order associations. Referring to Figure 2.11, we can see the overview of Hypergraph Transformer.

3. **SHT**: In existing GNN-based recommender systems, the primary approach is to iteratively perform message passing along user-item interaction edges to refine embeddings. While these models have shown effectiveness, they heavily rely on sufficient and high-quality training data to accurately capture user preferences. However, in many real-world recommendation scenarios, user behavior data tends to be noisy and exhibits skewed distributions, leading to suboptimal performance of GNN-based models in representing user preferences.

   To address these limitations, [49] introduces SHT, a novel Self-Supervised Hypergraph Transformer framework. SHT enhances user representations by explicitly exploring global collaborative relationships. Firstly, enhancing the graph neural collaborative filtering paradigm by incorporating a hypergraph transformer network to maintain global collaborative effects among users and items. This enables the capture of more comprehensive information.

Additionally, SHT introduces a cross-view generative self-supervised learning component for data augmentation over the user-item interaction graph. This further enhances the robustness of recommender systems by leveraging the distilled global context. It offers a promising approach to address the challenges of noisy and skewed user behavior data in recommender systems. The overview of SHT framework is shown in Figure 2.12.

Chapter 3

Tokenized Hypergraph Transformer(TokenHGT)

3.1 Tokenized Hypergraph Transformer(TokenHGT)

As mentioned earlier, modifying the structure of the transformer or adopting message passing methods can introduce certain limitations and challenges. Motivated by the success of TokenGT in addressing the limitations of graph-based models, we have incorporated similar principles in the development of our project in hypergraph area. By leveraging the insights gained from TokenGT, we aim to enhance the representation and processing of hypergraphs within the transformer framework. Therefore, we propose a straightforward approach of using the standard transformer to directly process hypergraphs.

In our approach, we consider each node and edge of the hypergraph as individual tokens. To capture their specific characteristics, we expand these tokens with appropriate token-wise embeddings, i.e. node identifiers and a special token. This enables us to encode rich information about the nodes and edges before feeding them into the standard transformer. The workflow of our approach is similar to that of the Vision Transformer (ViT)[11] model. We leverage the self-attention mechanism of the transformer to capture relationships and dependencies between the tokens within the hypergraph. The tokens are processed through multiple layers

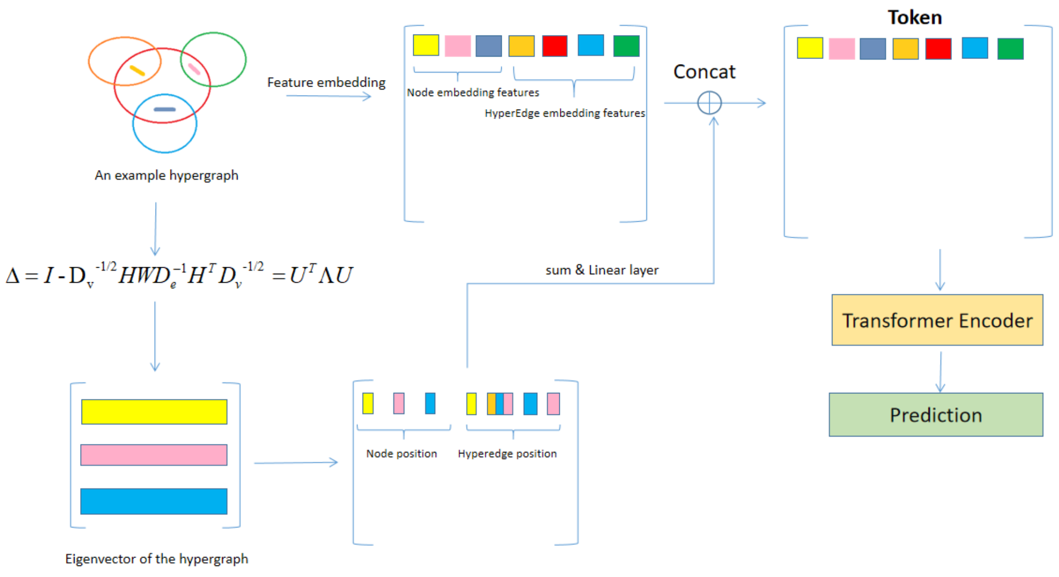


Figure 3.1. The overview of Tokenized Hypergraph Transformer(TokenHGT).

of self-attention and feed-forward neural networks, allowing the transformer to learn contextual representations for each token.

In this section, we introduce the Token Hypergraph Transformer (TokenHGT), a standard transformer architecture designed specifically for processing hypergraphs. TokenHGT leverages token-wise embeddings, which are composed of node identifiers and a special token, to capture the unique characteristics of nodes and hyperedges in the hypergraph. By representing each node and hyperedge as a token and incorporating node and a special token, TokenHGT enables the transformer to effectively model the complex relationships within the hypergraph. The node identifier captures the specific identity of each node, while the special token in tokenHGT serves as a representation of the entire hypergraph and captures global information that can be utilized for classification purposes.

Let $G = (\nu, \varepsilon)$ to present an input hypergraph, consists of a set of nodes ν and a set of hyperedges ε, which includes n nodes $\nu = \{v_1, ..., v_n\}$ and m hyperedges $\varepsilon = \{e_1, ..., e_m\}$, associated with features $X^\nu \in \mathbb{R}^{n \times C}$ and $X^\varepsilon \in \mathbb{R}^{m \times C}$. In TokenHGT, we adopt a token-based approach where each node and hyperedge is treated as an independent token. This results in a total of $(n+m)$ tokens. To construct the features of these tokens, we create a feature matrix X by concatenating the node features X^{ν} and hyperedge features X^{ε}.

$$X = [X^{\nu}, X^{\varepsilon}] \tag{3.1}$$

The feature matrix $X \in \mathbb{R}^{(n+m)\times C}$, $[\cdot, \cdot]$ is the concatenation operator.

A straightforward approach to processing a graph is to directly input the tokens X into a Transformer. However, this method overlooks the graph connectivity information, which is crucial for capturing the structural relationships within the graph. In our study, we consider this approach as an ablation experiment to compare its performance with TokenHGT(lap). The results of the ablation experiment are presented in Table 4.2.

3.1.1 Node Identifier

As mentioned earlier, the first component of the token-wise embedding in our approach is the node identifier. This component plays a crucial role in representing the connectivity structure inherent in the input hypergraph.

For a given input hypergraph $G = (\nu, \varepsilon)$, we generate n node-wise Laplacian eigenvectors $P \in \mathbb{R}^{n\times d_p}$, which we refer to as node identifiers, we can obtain the Laplacian eigenvectors by applying the formula 2.6 and 2.7, which allows us to compute the eigenvalues and eigenvectors of the Laplacian matrix associated with the input hypergraph. The eigenvector matrix has a shape of $n \times n$, where n represents the number of nodes in the hypergraph. Each eigenvector corresponds to a specific node. To process the eigenvectors, we sort the eigenvalues in ascending order and rearrange the eigenvectors accordingly. Finally, we select the top-**dp** length of each eigenvector as Laplacian eigenvector matrix P, retaining the most informative components.

Next, we augment the feature matrix with the Laplacian matrix in the following manner:

• For each node $v \in \nu$ corresponds to an eigenvector $P_{v_i} \in P$, we take those eigenvectors $P_v = [P_{v_1}, ..., P_{v_n}] \in \mathbb{R}^{n \times C}$.

• For each hyperedges $e \in \varepsilon$ corresponding to at least two nodes $\{v_x, ..., v_z\} \in e_i$, so we take sum of those eigenvectors $P_e = [sum(P_{e_1}), ..., sum(P_{e_n})] \in \mathbb{R}^{m \times C}$ for capturing the collective information of the nodes within the hyperedge. By summing the eigenvectors, we aim to enhance the representation of the hyperedges and incorporate the shared characteristics of the nodes involved in the hyperedge.

Finally, we perform feature fusion by concatenating these feature matrices to create the input tokens X. This process combines the node-wise and edge-wise representations into a single input representation, allowing for comprehensive information integration. The concatenation of these feature matrices serves as the input tokens for further processing in the tokenHGT model.

$$X = [X^{\nu}, X^{\varepsilon}, P_v, P_e] \tag{3.2}$$

The feature matrix $X \in \mathbb{R}^{(n+m) \times 2C}$, $[\cdot, \cdot]$ is the concatenation operator.

3.1.2 Special Token

To account for the entire hypergraph as a whole and enable classification tasks, we introduce a special graph token in TokenHGT. It's inspired by approaches such as BERT and ViT, the [graph] token is further projected using a trainable embedding weight matrix $\omega \in \mathbb{R}^{1 \times C}$. The final input matrix $X \in \mathbb{R}^{(n+m+1) \times 2C}$ to the standard transformer is constructed as follows:

$$X = [X^{\nu}, X^{\varepsilon}, P_v, P_e, \omega] \tag{3.3}$$

This graph token serves as a representation of the entire hypergraph and captures global information that can be utilized for classification purposes. By including the graph token in the token-wise embeddings, TokenHGT gains the ability to consider the overall structure and characteristics of the hypergraph when making predictions. This allows the model to incorporate global context and leverage information from the entire hypergraph, enhancing its classification capabilities.

The graph token acts as a summary of the hypergraph and is updated during the transformer's self-attention mechanism, enabling it to capture relevant information and influence the model's decision-making process. By incorporating the graph token, TokenHGT achieves a holistic understanding of the hypergraph, which is particularly beneficial for classification tasks where the entire hypergraph's context is important.

3.1.3 Main Transformer

After obtaining the augmented token features using node identifiers and the special graph token, we input them into the main encoder, which is the standard transformer.

In the context of classification tasks, we extend the tokenHGT model by incorporating an MLP (Multi-Layer Perceptron) at the end of the model. This MLP serves as a classifier, taking the encoded token representations as input and producing class predictions. The architecture of the MLP can be customized and adjusted to suit different classification objectives or downstream tasks. This flexibility allows the tokenHGT model to be easily adapted and fine-tuned for various applications beyond classification.

3.2 Implementation Details

3.2.1 Algorithm details

Let us delve into the details of tokenHGT. The code is written entirely in PyTorch.

1. Processing Node & Hyperedge features

Firstly, it is common for the dimensions of node features and hyperedge features to be different. For example, Let node features be $X^{\nu} \in \mathbb{R}^{n \times C_1}$ and hyperedge features be $X^{\varepsilon} \in \mathbb{R}^{m \times C_2}$.

To address this, we employ two separate embedding layers to process each feature type independently, ensuring that they are transformed into the same dimension. This process is similar to the embedding of words in natural language processing tasks, where words are mapped to fixed-dimensional vector representations. After applying the embedding layer, we perform a summation operation along the middle dimension. As a result, for a batch of data, the dimension of the node feature matrix transforms from $[sum(node\_num), C_1]$ to $[sum(node\_num), hidden\_dim]$, i.e. the node features are projected from $\mathbb{R}^{n \times C_1}$ to $\mathbb{R}^{n \times C}$. We apply the same method to project the hyperedge features, ensuring that their dimensions are the same.

2. Processing Laplacian features

For a batch of Laplacian eigenvectors, we first select the top-**dp** eigenvectors. If an eigenvector's length exceeds the dp length, we truncate it. Conversely, if an eigenvector's length is less than the dp length, we pad it to reach the dp length.

Next, we collect and organize the eigenvectors as described in Chapter 3.1.1. To align the dimensions between the eigenvectors $[P_v, P_e] \in \mathbb{R}^{(n+m) \times dp}$ and the hidden dimension, we employ a linear layer to project the eigenvector dimension to the desired hidden dimension.

3.2.2 Model details

In this chapter, we provide a detailed overview of the model configuration, including the optimizer, loss function, and other important components utilized in the training and evaluation processes.

- **Optimizer**:

  The model was trained using the Stochastic Gradient Descent(SGD) optimizer. SGD is a widely used optimization algorithm in machine learning and deep learning. It is particularly effective in training neural network models. The goal of SGD is to iteratively update the model's parameters in order to minimize a given loss function.

  The core idea behind SGD is to compute the gradients of the model's parameters with respect to the loss function using a subset of the training data, known as a mini-batch. Instead of considering the entire training dataset, SGD randomly samples mini-batches, which introduces randomness into the parameter updates. This random sampling helps avoid getting stuck in local minima and enables faster convergence. The learning rate controls the step size of the parameter updates and is a crucial hyperparameter to be tuned. It determines how quickly the model adapts to the training data and affects the convergence speed and stability. However, SGD can be sensitive to the choice of learning rate and may require careful tuning to achieve optimal performance.

- **Loss Function**:

  The chosen loss function for the model is Cross Entropy Loss, it is a commonly used loss function in classification tasks, particularly in deep learning. It measures the dissimilarity between the predicted probability distribution and the true distribution of the target labels.

  In classification problems, the model predicts the probability distribution over multiple classes for each input sample. The Cross Entropy Loss quantifies how well the predicted probabilities align with the actual labels. It is derived from the concept of information theory, specifically the measure of entropy. Mathematically, given a predicted probability distribution p and the true distribution y, the Cross Entropy Loss is computed as:

$$L_{CE} = -\sum_{i=1}^{n} y_i log(p_i) \tag{3.4}$$

where y_i represents the true probability of the i-th class and p_i represents the predicted probability of the i-th class.

The Cross Entropy Loss encourages the model to assign higher probabilities to the correct class labels. It penalizes large differences between the predicted probabilities and the true probabilities, leading to more accurate and confident predictions. The negative sign in the formula ensures that the loss value is positive and increases as the predictions deviate further from the true labels. The use of Cross Entropy Loss is prevalent in many deep learning architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for various classification tasks like image recognition, sentiment analysis, and natural language processing.

- **Learning Rate Scheduler**:

  A common technique to reduce the learning rate step-by-step during training is to use a callback function. The basic idea of a learning rate callback is to define a function that monitors the training progress and adjusts the learning rate accordingly at specific intervals or conditions. This allows for dynamic learning rate scheduling based on the model's performance or the number of training iterations/epochs.

- **Early stopping**:

  Early stopping is a technique used in machine learning to prevent overfitting and improve generalization performance. It involves monitoring the model's performance on a validation set during the training process and stopping the training early when the model's performance on the validation set starts to deteriorate.

The basic idea behind early stopping is that as the model continues to train, it may start to overfit the training data, leading to a decrease in performance on unseen data. By monitoring the performance on the validation set, we can detect when this overfitting starts to occur. The validation set is a separate dataset that is not used for training but is used to assess the model's performance during training. It typically involves tracking a metric, such as validation loss or accuracy, and comparing it to the best observed value. If the metric does not improve for a certain number of consecutive epochs, the training process is halted, and the model with the best observed performance on the validation set is selected as the final model.

Chapter 4

Experiment

4.1 Evaluation dataset

The tokenHGT algorithm is designed to operate at the graph level, making it suitable for datasets that contain a significant number of hypergraphs. Ideally, the dataset should include both node and hyperedge features to capture the structural and attribute information inherent in the hypergraphs.

However, it is challenging to find readily available datasets that meet these requirements. Therefore, we have explored two methods to create suitable hypergraph datasets.

The first method is called Dual Hypergraph Transformation (DHT)[24], which involves transforming existing graph datasets into hypergraph representation. This transformation aims to capture the higher-order relationships among nodes and provide a more comprehensive view of the data. The DHT algorithm, as shown in Figure 4.1

The second method involves using Latent Dirichlet Allocation (LDA)[3] to convert text documents into hypergraphs[9]. LDA is a topic modeling technique that assigns topics to words in a document. By treating each topic as a hyperedge and words as nodes, a hypergraph can be constructed from the document, capturing the

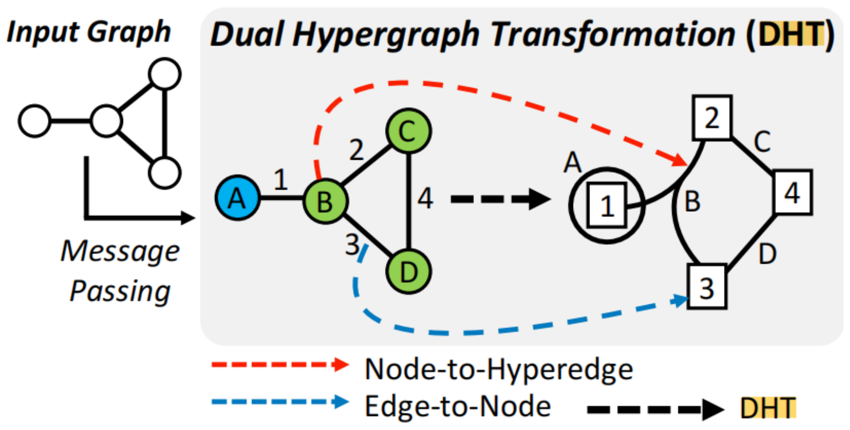


Figure 4.1. DHT is employed to convert the input graph to hypergraph.

underlying semantic relationships.

Both of these methods offer viable solutions for generating hypergraph datasets suitable for training and evaluating tokenHGT. However, for our specific use case, we have chosen to employ LDA to convert text documents into hypergraphs. This decision was made based on the availability and suitability of the text dataset at hand.

In order to ensure a thorough and unbiased evaluation, our experiments are conducted on five benchmark datasets that cover a range of different domains. These datasets include 20-Newsgroups (20NG)[1], Reuters (R8 and R52), Ohsumed, and Movie Review (MR).

1. **20-Newsgroups (20NG)**: This dataset consists of newsgroup posts classified into 20 different topics, making it a widely used benchmark for text classification tasks.
2. **Reuters (R8 and R52)**: The Reuters dataset comprises news articles categorized into different topics. We have utilized two variants of this dataset: R8, which contains eight main categories, and R52, which contains 52 categories.
3. **Ohsumed**: The Ohsumed dataset focuses on medical text classification, containing abstracts from medical literature classified into 23 topics. It serves

Table 4.1. Summary statistics of the evaluation datasets.

Dataset	**20NG**	**R8**	**R52**	**Ohsumed**	**MR**
Docs	18846	7674	9100	7400	10662
Train	11314	5485	6532	3357	7108
Test	7532	2189	2568	4043	3554
Words	42757	7688	8892	14157	18764
Average Length	221.26	65.72	69.82	135.82	20.39
Classes	20	8	52	23	2

as a valuable resource for evaluating algorithms in the healthcare domain.

4. **Movie Review (MR)**: The Movie Review dataset consists of movie reviews labeled as either positive or negative sentiment. It is commonly used for sentiment analysis tasks and provides a good benchmark for text classification algorithms.

The details of the benchmark datasets used in our experiments are summarized in Table 4.1. This table provides an overview of the key characteristics and statistics of each dataset, including the number of documents, the number of train and test dataset, the number of total words, the average document length, and the number of classes. These datasets are representative of various domains and provide a diverse range of text samples for training and evaluation. The number of classes varies across datasets, allowing us to assess the model's performance in both binary and multi-class classification scenarios. The average document length provides an indication of the text complexity and the level of information contained in each sample.

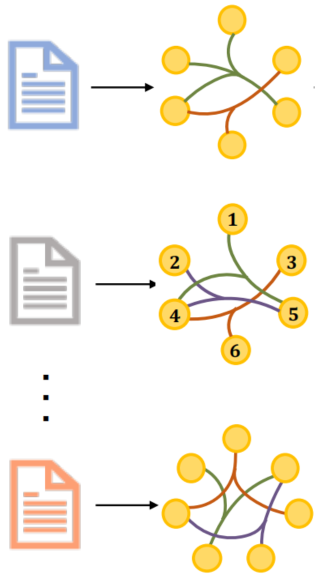


Figure 4.2. Convert text documents into hypergraphs.

4.2 LDA-Hypergragh

To enhance the semantic context of each word, [9] incorporate semantic hyperedges that capture topic-related high-order correlations between words[32]. This approach allows us to explore the relationships between words in a text document beyond direct connections. Specifically, they employ Latent Dirichlet Allocation (LDA)[3] to extract latent topics T from the text documents. Each topic $t_i = (\theta_1, ..., \theta_w)$ represents a probability distribution over the vocabulary, where w denotes the vocabulary size.

To construct semantic hyperedges, the authors consider each topic as a distinct hyperedge that connects the top K words with the highest probabilities in the document. This means that for each topic, selecting the K most representative words based on their probabilities in that topic. By incorporating these topic-related hyperedges, this method can enrich the high-order semantic context of words within each document. This approach allows to capture the underlying semantic relationships between words and leverage them for improved text understanding and analysis.

4.3 Deep Text Classification

Deep text classification refers to the application of deep learning techniques for the task of classifying textual data into predefined categories or classes. It involves training deep neural networks to automatically learn hierarchical representations of text, capturing both local and global dependencies within the data.

Deep text classification models are designed to handle the inherent complexities and nuances of natural language. They are capable of automatically extracting informative features from raw text data, eliminating the need for manual feature engineering. This makes them highly flexible and adaptable to a wide range of text classification tasks, including sentiment analysis, document categorization, spam detection, and many others. One of the key advantages of deep text classification models is their ability to capture semantic and contextual information from text. By leveraging deep neural network architectures, such as convolutional neural networks (CNNs)[30] or recurrent neural networks (RNNs)[38], these models can effectively learn representations that capture the underlying meaning and structure of text. This enables them to achieve superior performance compared to traditional machine learning approaches.

Here are some popular models for text classification, along with their corresponding accuracy results as shown in Table 4.2:

1. **CNN-rand**: CNNs, or Convolutional Neural Networks, have been widely applied to text classification tasks with great success. Originally designed for image analysis, CNNs have shown their effectiveness in learning hierarchical representations from sequential data, making them suitable for text analysis as well. In the context of text classification, CNNs operate on textual inputs represented as sequences of words or characters. They use convolutional filters to capture local patterns or n-grams in the input text. These filters slide over the input, applying a set of convolutional operations to capture different features at various levels of granularity.

By applying multiple filters with different sizes, CNNs can capture various patterns and features present in the text. The outputs of the filters are then passed through non-linear activation functions, such as ReLU, to introduce non-linearities and enhance the model's expressive power. After the convolutional layers, CNNs typically employ pooling operations, such as max pooling or average pooling, to reduce the dimensionality of the extracted features and capture the most salient information. This helps in identifying important features and reducing noise in the data. Finally, the extracted features are fed into fully connected layers for classification, where the model learns to map the input features to the corresponding class labels. These layers can be followed by softmax activation to generate probability scores for each class.

CNNs for text classification have proven to be effective in capturing local dependencies, detecting important patterns, and generalizing well to unseen data. They are particularly useful when the position of words or characters in the text plays a crucial role in determining the class label.

2. **LSTM**: Long Short-Term Memory[20], is a type of recurrent neural network (RNN) architecture that has shown promising results in text classification tasks. Unlike traditional feed-forward neural networks, LSTM is capable of capturing long-range dependencies and modeling sequential information effectively. The LSTM cell structure is depicted in Figure 4.3.

   In the context of text classification, LSTM operates on sequential inputs, such as sentences or documents. It processes the input tokens one at a time, updating its internal state based on the current input and the information it has learned from the previous tokens. This enables the model to consider the context and sequential relationships between words or characters in the text. The key component of LSTM is its memory cell, which allows the model to selectively remember or forget information over time. This mechanism

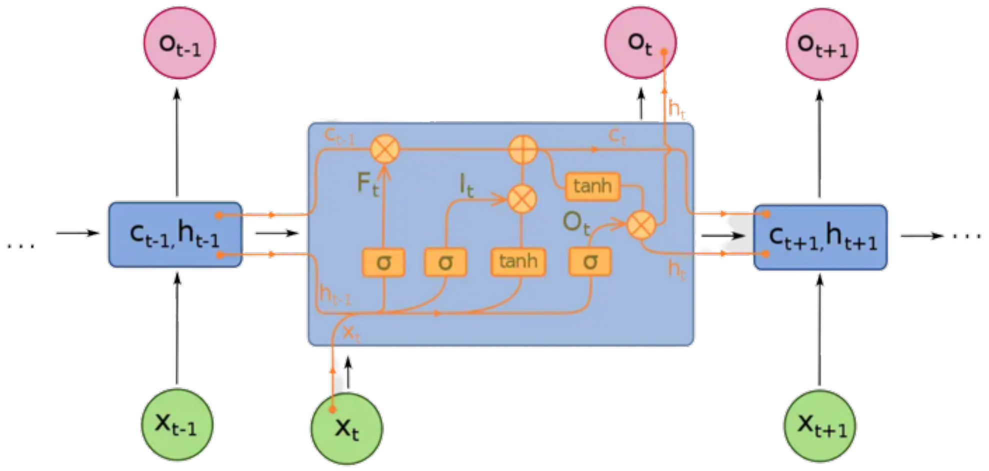


Figure 4.3. The structure of LSTM cell.

helps LSTM in capturing important information from earlier parts of the sequence and retaining it for longer durations. It mitigates the vanishing gradient problem often encountered in traditional RNNs, allowing LSTM to handle longer sequences and preserve relevant context. It also includes gates, such as the input gate, forget gate, and output gate, which regulate the flow of information through the memory cell. These gates control the update of the internal state and determine how much information should be stored or discarded at each time step.

By learning from the sequential data, LSTM can capture semantic and syntactic dependencies in the text, making it suitable for tasks like sentiment analysis, document classification, and text generation. Its ability to model long-range dependencies and handle varying-length sequences has contributed to its popularity in the field of natural language processing (NLP).

3. **Bi-LSTM**: short for Bidirectional Long Short-Term Memory[23], is a variant of the LSTM architecture that incorporates information from both past and future contexts in text classification tasks. While traditional LSTM processes the input sequence in a forward direction, Bi-LSTM enhances the model's ability to capture context by processing the sequence in both forward and backward directions simultaneously. Figure 4.4 illustrates a Bi-LSTM model.

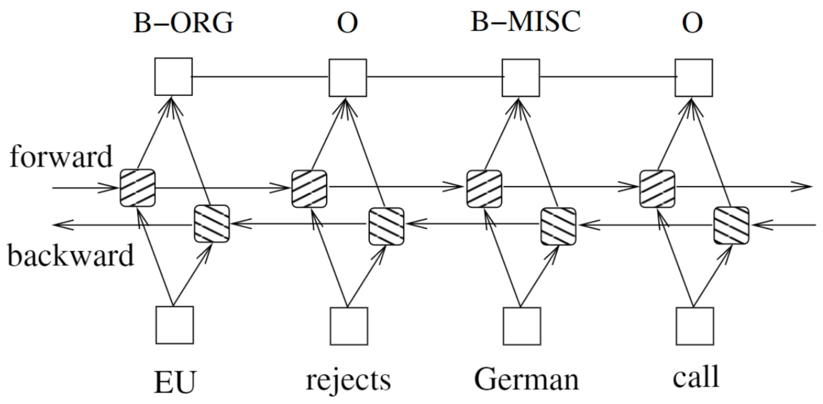


Figure 4.4. A Bi-LSTM model.

By utilizing two separate LSTM layers, one processing the sequence in the forward direction and the other in the backward direction, Bi-LSTM can capture information from preceding and succeeding tokens at each time step. This bidirectional approach allows the model to have a more comprehensive understanding of the input sequence and capture dependencies from both past and future contexts. Bi-LSTM is particularly effective in scenarios where understanding the surrounding context is crucial for accurate classification, such as sentiment analysis or named entity recognition. By considering the complete context, the model can better grasp the relationships between words or characters and make more informed predictions. The output of Bi-LSTM is typically obtained by concatenating the hidden states from both forward and backward LSTM layers. This combined representation contains information from both directions and can be further processed by additional layers or fed directly into a classifier for text classification.

Overall, Bi-LSTM has proven to be a powerful architecture for text classification tasks, leveraging the advantages of bidirectionality to capture context and improve the model's performance in understanding and classifying textual data.

4. **SWEM**: Simple Word-Embedding Model[40] is a simple yet effective approach

for text classification tasks. It represents a document or sentence by directly aggregating the word embeddings of its constituent words. Instead of relying on complex recurrent or convolutional neural network architectures, SWEM applies simple pooling operations to capture the essence of the text.

The most common pooling operations used in SWEM are max pooling and average pooling. Max pooling selects the maximum value from each word embedding dimension, resulting in a fixed-length representation. Average pooling, on the other hand, calculates the mean value across each word embedding dimension. These pooling operations effectively summarize the information contained in the word embeddings, capturing important patterns and features of the text. SWEM's simplicity makes it computationally efficient and easy to implement. It has shown promising results in various text classification tasks, especially in scenarios where capturing local word order or long-range dependencies is less important. However, it's worth noting that SWEM's pooling-based approach may overlook the sequential nature of text and discard fine-grained information about word order. As a result, it may be less suitable for tasks that heavily rely on word order, such as sentiment analysis in cases where negations play a crucial role.

Nonetheless, SWEM remains a valuable and efficient option for text classification, providing a straightforward approach for representing documents or sentences using word embeddings and simple pooling operations.

5. **LEAM**: Label-Embedding Attentive Model[46] is a text classification model that leverages label embeddings to improve the performance of traditional deep learning models. It incorporates label information into the learning process by embedding the labels in a low-dimensional space. The core joint embedding method is shown in Figure 4.5.

   In LEAM, the label embeddings serve as additional context for the model

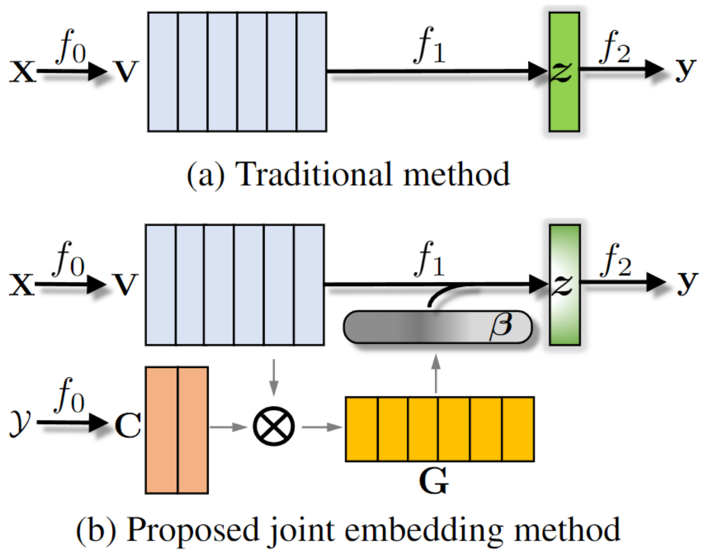


Figure 4.5. (a). The traditional NLP method. (b). The proposed joint embedding method.

to better understand the relationships between different classes. The model uses attention mechanisms to dynamically weigh the importance of each label embedding based on the input text. By attending to relevant label embeddings, LEAM can focus on the most informative aspects of the text for classification. This approach helps address the issue of data sparsity and provides a mechanism for the model to learn from limited labeled examples. By incorporating label embeddings and leveraging attention, LEAM enhances the discriminative power of the model and improves its ability to classify text accurately.

LEAM has demonstrated promising results in various text classification tasks, showing improvements over traditional deep learning models that do not consider label embeddings. It offers a way to effectively incorporate label information into the learning process and boost the performance of text classification models.

6. **fastText**: fastText[25] is a popular text classification model that is known for its efficiency and effectiveness in handling large-scale text datasets. It is based on the concept of word embeddings and uses a shallow neural network architecture. The structure of fastText is shown in Figure 4.6.

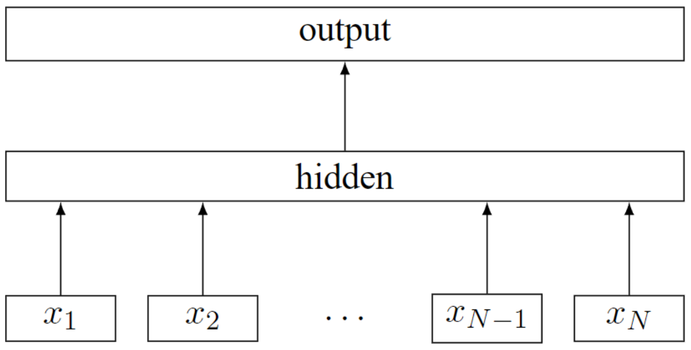


Figure 4.6. The model architecture of fastText for a sentence with N ngram features $x_1, ..., x_N$.

One key feature of fastText is its ability to capture sub-word information, which allows it to handle out-of-vocabulary words and better represent the semantics of rare words. It achieves this by representing each word as a combination of character n-grams, which are sub-sequences of characters within the word. With its efficient training algorithm and compact representations, fastText is capable of classifying text at high speeds, making it suitable for real-time or large-scale text classification tasks. It has been widely adopted in various applications and has shown competitive performance compared to more complex deep learning models.

7. **Graph-CNN**: Graph Convolutional Neural Network[8] is a text classification model that incorporates graph-based representations for capturing the structural information of text data. It leverages the idea of graph convolution to exploit the relationships between words in a document. The structure of CNN on graph is shown in Figure 4.7.

   In Graph-CNN, each word in the document is represented as a node in a graph, and the connections between words are defined by their semantic relationships. By applying graph convolutional operations, the model can aggregate information from neighboring words to enhance the representation of each word. The

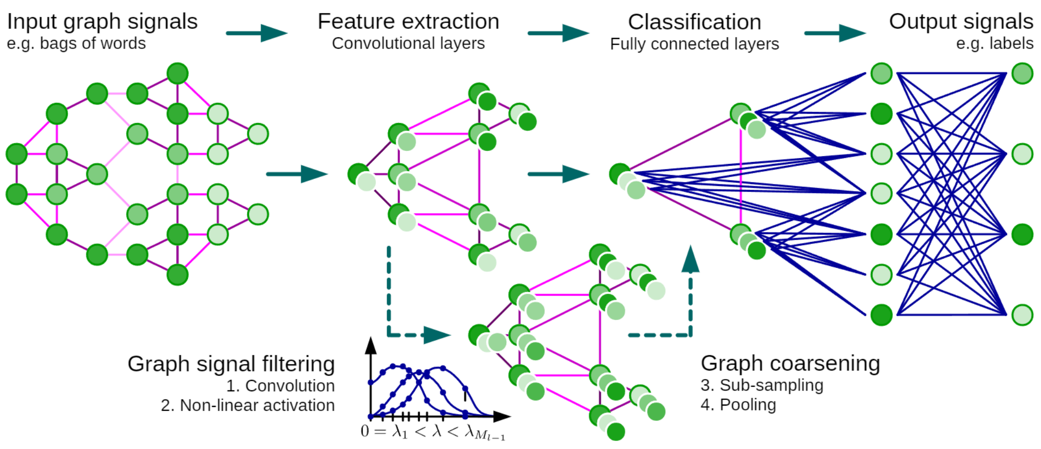


Figure 4.7. Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

use of graph-based representations in Graph-CNN allows the model to capture contextual dependencies and capture the global structure of the text. It has been shown to be effective in handling text classification tasks, especially when dealing with documents that exhibit complex semantic relationships.

Graph-CNN provides a promising approach to text classification by incorporating graph structures and leveraging graph convolutional operations. By considering the connectivity of words, it can capture richer contextual information and improve the accuracy of text classification tasks.

8. **TextGCN**: Text Graph Convolutional Network[51] is a text classification model that utilizes graph convolutional networks to capture the semantic relationships between words in a document. It represents the text data as a graph, where words are nodes and their co-occurrence or co-occurrence patterns are used to define the edges. Figure 4.8 illustrates a schematic of Text GCN with an example taken from the Ohsumed corpus.

   TextGCN leverages graph convolutional operations to propagate information across the graph, allowing it to capture the contextual dependencies and semantic connections between words. By aggregating information from neighboring

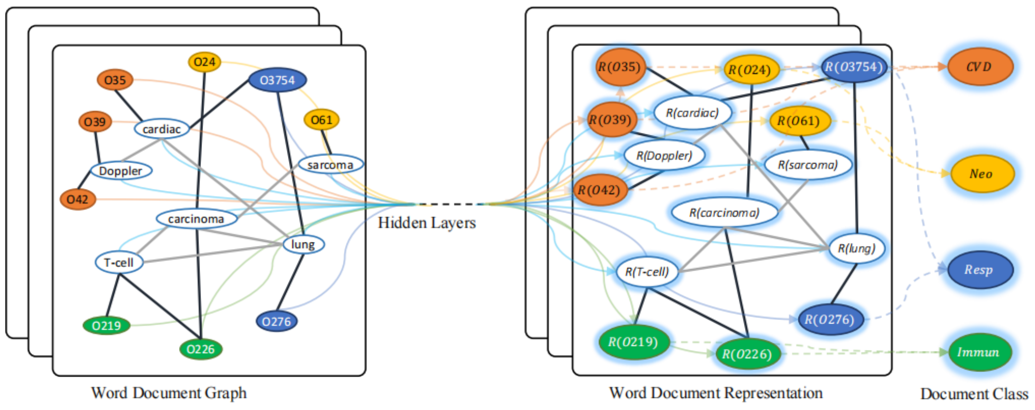


Figure 4.8. A schematic of Text GCN with an example taken from the Ohsumed corpus.

words, the model enhances the representation of each word and improves the overall classification performance.

The incorporation of graph convolutional networks in TextGCN enables it to effectively capture the structural information of text data, making it suitable for text classification tasks. It has demonstrated promising results in various applications, showcasing its ability to leverage the underlying graph structure for improved classification accuracy.

9. **Text-level GNN**: Text-level GNN[21] is a graph neural network (GNN)-based model for text classification. Unlike previous GNN-based methods, which use a fixed corpus-level graph structure, Text-level GNN builds individual graphs for each input text. This approach allows for online testing and reduces memory consumption. By sharing global parameters, the model preserves global information while removing the dependence between individual texts and the entire corpus. Additionally, Text-level GNN constructs graphs using smaller windows in the text, extracting more local features and reducing edge numbers and memory usage. Experimental results demonstrate that Text-level GNN outperforms existing GCN approaches on various text classification datasets while consuming less memory.

4.4 Hyper-parameters

The following hyper-parameters were considered and carefully chosen to ensure the optimal performance of the proposed models. Each hyper-parameter plays a significant role in shaping the behavior and outcomes of the experiments. The values were determined through extensive experimentation and empirical analysis. The device is one NVIDIA GeForce RTX 2060 GPU.

1. **Learning Rate**
   (a) Value: 0.01
   (b) Description: The chosen initial learning rate value of 0.01 strikes a balance between convergence speed and stability. The step size is 15 represents the number of epochs after which the learning rate is adjusted, while the gamma value controls the extent of the adjustment. In this configuration, the learning rate will be multiplied by 0.1 every 15 epochs.

2. **Batch size**
   (a) Value: 8
   (b) Description: The batch size refers to the number of training examples used in a single iteration of the training algorithm. A batch size of 8 balances the computational efficiency of larger batch sizes with the regularization benefits of smaller batch sizes. It allows for efficient training while still introducing enough randomness to promote generalization.

3. **Number of Hidden Units**
   (a) Value: 308
   (b) Description: Setting the number of hidden units to 308 implies that the model will have 308 hidden units in a particular layer or component. In this case, the hidden dimension of the Laplacian eigenvectors is 300. By concatenating it with the number of hidden units (308), the resulting size would be 608. The fact that 608 is evenly divisible by 8 (the number of attention heads) is advantageous in terms of computational efficiency.

4. **Dp**
   (a) Value: 20
   (b) Description: Setting dp to 20 means that selecting the eigenvectors corresponding to the top-dp minimum eigenvalues. It's important to note that the choice of dp=20 may vary depending on the specific dataset, problem domain, or application. The selection of dp=20 is based on empirical evidence and the specific requirements of your experiment or analysis.
5. **Dropout Rate**
   (a) Value: 0.15
   (b) Description: Based on multiple experiments, it has been observed that setting the dropout rate to 0.15 yields better performance compared to the default value of 0.1 in the standard transformer model. Increasing the dropout rate beyond 0.15 has shown to have a negative impact on the model's performance.
6. **Number of Attention Head**
   (a) Value: 8
   (b) Description: Setting the number of attention heads in the transformer model to 8 produces the same result as using 16 attention heads from many experiments, and it will reduce the computational complexity.
7. **Number of Transformer Encoder Layer**
   (a) Value: 2
   (b) Description: Setting the number of transformer encoder layers to 2 prevents overfitting and produces better results compared to using a higher number of layers.
8. **Epochs**
   (a) Value: 100

(b) Description: I have set the number of epochs to 100 for training the model and it will stop early to avoid overfitting.

4.5 Accuracy and Compared methods

The table 4.2 presents the test accuracy results of various models on the document classification task. From the result, graph-based models, also known as message-passing-based models, typically outperform sequence-based models and word embedding-based models in terms of performance.

And the tokenHGT(lap) model achieves competitive performance in document classification tasks. It outperforms several traditional models such as CNN-rand, LSTM, SWEM, and LEAM, with consistently high accuracy scores across all three datasets (R8, R52, and MR). The tokenHGT(lap) model demonstrates its effectiveness in leveraging graph-based techniques to capture and utilize the structural information within the text data. It achieves a remarkable accuracy of 0.9612 on R8, 0.9146 on R52, and an impressive 0.7780 on the MR dataset, which also beat all the massage-passing methods. These results highlight the efficacy of the tokenHGT approach for processing hypergraph tasks.

Table 4.2. Test Accuracy on document classification task. The data referenced in this study is sourced from [9], and all values presented are averaged.

Model	**R8**	**R52**	**MR**
CNN-rand	0.9402	0.8537	0.7498
LSTM	0.9368	0.8554	0.7506
Bi-LSTM	0.9631	0.9054	0.7768
SWEM	0.9532	0.9294	0.7665
LEAM	0.9331	0.9184	0.7695
fastText(bigrams)	0.9474	0.9099	0.7624
Graph-CNN	0.9699	0.9275	0.7722
TextGCN (inductive)	0.9578	0.8820	0.7480
TextGCN (transductive)	0.9707	0.9356	0.7674
Text-level GNN	**0.9789**	**0.9460**	0.7547
transformer	0.9553	0.9049	0.7705
tokenHGT(lap)	0.9612	0.9146	**0.7780**

Chapter 5

Conclusions

In conclusion, our tokenHGT model has shown promising performance on the R8 and R52 datasets, particularly outperforming the TextGCN (inductive)[51][9] model. Furthermore, tokenHGT has shown better results compared to all Graph-Convolution-based (or message passing) models on the MR dataset. This findings align with our expectations, as they validate the notion that message passing methods can introduce certain limitations, which the pure transformer architecture of tokenHGT effectively addresses.

However, an important limitation of our algorithm that did not fully manifest in the results table: tokenHGT is not good at processing large hypergraphs. Our experiments on the 20newsgroups[1] and Ohsumed datasets yielded subpar performances. According to Graphomer[53], the self-attention module exhibits a quadratic complexity, which poses limitations on its applicability to large graphs. Our experimental results support this opinion, as we observed a correlation between the average text length and the size of the hypergraph, impacting the transformer's performance. This limitation provides a potential explanation for tokenHGT's varying performance across different datasets. Another limitation of this algorithm is that it requires a suitable hypergraph dataset, which can be challenging to find. All of these limitations could be our future work.

In summary, our tokenHGT model has demonstrated its effectiveness in overcoming the limitations of message passing methods, leading to superior performance on specific datasets. Meanwhile, the pure transformer architecture guarantees the versatility of the models. The correlation between graph size and transformer performance further sheds light on the strengths and limitations of the tokenHGT framework. These insights enhance our understanding of tokenHGT's applicability and performance in diverse graph analysis tasks.

Bibliography

[1] 20 newsgroups dataset. http://qwone.com/~jason/20Newsgroups/. Accessed on 2023.

[2] Gema Bello-Orgaz and David Camacho. Evolutionary clustering algorithm for community detection using graph-based information. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 930–937. IEEE, 2014.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

[5] Hanxuan Cai, Huimin Zhang, Duancheng Zhao, Jingxing Wu, and Ling Wang. Fp-gnn: a versatile deep learning architecture for enhanced molecular property prediction. *Briefings in Bioinformatics*, 23(6):bbac408, 2022.

[6] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[7] Zahra Zamanzadeh Darban and Mohammad Hadi Valipour. Ghrs: Graph-based hybrid recommendation system with application to movie recommendation. *Expert Systems with Applications*, 200:116850, 2022.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[9] Kaize Ding, Jianling Wang, Jundong Li, Dingcheng Li, and Huan Liu. Be more with less: Hypergraph attention networks for inductive text classification. *arXiv preprint arXiv:2011.00387*, 2020.

[10] Ronky Francis Doh, Conghua Zhou, John Kingsley Arthur, Isaac Tawiah, and Benjamin Doh. A systematic review of deep knowledge graph-based recommender systems, with focus on explainable embeddings. *Data*, 7(7):94, 2022.

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Anastasios Drosou, Ilias Kalamaras, Stavros Papadopoulos, and Dimitrios Tzovaras. An enhanced graph analytics platform (gap) providing insight in big network data. *Journal of Innovation in Digital Ecosystems*, 3(2):83–97, 2016.

[13] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[14] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. 2020.

[15] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019.

[16] Laurent Galluccio, Olivier Michel, Pierre Comon, and Alfred O Hero III. Graph based k-means clustering. *Signal Processing*, 92(9):1970–1984, 2012.

[17] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1623–1625, 2022.

[18] Frauke Heinzle, Karl-Heinrich Anders, and Monika Sester. Graph based approaches for recognition of patterns and implicit information in road networks.

In *Proceedings of the 22nd international cartographic conference*, pages 11–16. ICA Washington, DC, 2005.

[19] Yu-Jung Heo, Eun-Sol Kim, Woo Suk Choi, and Byoung-Tak Zhang. Hypergraph transformer: Weakly-supervised multi-hop reasoning for knowledge-based visual question answering. *arXiv preprint arXiv:2204.10448*, 2022.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Text level graph neural network for text classification. *arXiv preprint arXiv:1910.02356*, 2019.

[22] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 65–73, 2002.

[23] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[24] Jaehyeong Jo, Jinheon Baek, Seul Lee, Dongki Kim, Minki Kang, and Sung Ju Hwang. Edge representation learning with hypergraphs. *Advances in Neural Information Processing Systems*, 34:7534–7546, 2021.

[25] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[26] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022.

[27] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[28] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[29] Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semi-supervised graph clustering: a kernel approach. In *Proceedings of the 22nd international conference on machine learning*, pages 457–464, 2005.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[32] Hu Linmei, Tianchi Yang, Chuan Shi, Houye Ji, and Xiaoli Li. Heterogeneous graph attention networks for semi-supervised short text classification. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 4821–4830, 2019.

[33] Yujia Liu, Kang Zeng, Haiyang Wang, Xin Song, and Bin Zhou. Content matters: a gnn-based model combined with text semantics for social network cascade prediction. In *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I*, pages 728–740. Springer, 2021.

[34] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems*, 214:106746, 2021.

[35] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H Pham, Jafar M Al-Kofahi, and Tien N Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, pages 383–392, 2009.

[36] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

[37] P Pradhyumna, GP Shreya, et al. Graph neural network (gnn) in image and video understanding using deep learning for computer vision applications. In *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 1183–1189. IEEE, 2021.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[39] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[40] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *arXiv preprint arXiv:1805.09843*, 2018.

[41] Daniel Spielman. Spectral graph theory. *Combinatorial scientific computing*, 18:18, 2012.

[42] Guang Tan, Marin Bertier, and A-M Kermarrec. Visibility-graph-based shortest-path geographic routing in sensor networks. In *IEEE INFOCOM 2009*, pages 1719–1727. IEEE, 2009.

[43] Siliang Tang, Wenqiao Zhang, Zongshen Mu, Kai Shen, Juncheng Li, Jiacheng Li, and Lingfei Wu. Graph neural networks in computer vision. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 447–462, 2022.

[44] Khushboo S Thakkar, Rajiv V Dharaskar, and MB Chandak. Graph-based algorithms for text summarization. In *2010 3rd International Conference on Emerging Trends in Engineering and Technology*, pages 516–519. IEEE, 2010.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] Guoyin Wang, Chunyuan Li, Wenlin Wang, Yizhe Zhang, Dinghan Shen, Xinyuan Zhang, Ricardo Henao, and Lawrence Carin. Joint embedding of words and labels for text classification. *arXiv preprint arXiv:1805.04174*, 2018.

[47] Hao Wang, Fanjiang Xu, Xiaohui Hu, and Yukio Ohsawa. Ideagraph: a graph-based algorithm of mining latent information for human cognition. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 952–957. IEEE, 2013.

[48] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

[49] Lianghao Xia, Chao Huang, and Chuxu Zhang. Self-supervised hypergraph transformer for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2100–2109, 2022.

[50] Gui Yang, Wenping Zheng, Chenhao Che, and Wenjian Wang. Graph-based label propagation algorithm for community detection. *International Journal of Machine Learning and Cybernetics*, 11:1319–1329, 2020.

[51] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[52] Shuqian Ye, Jiechun Liang, Rulin Liu, and Xi Zhu. Symmetrical graph neural network for quantum chemistry with dual real and momenta space. *The Journal of Physical Chemistry A*, 124(34):6945–6953, 2020.

[53] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.